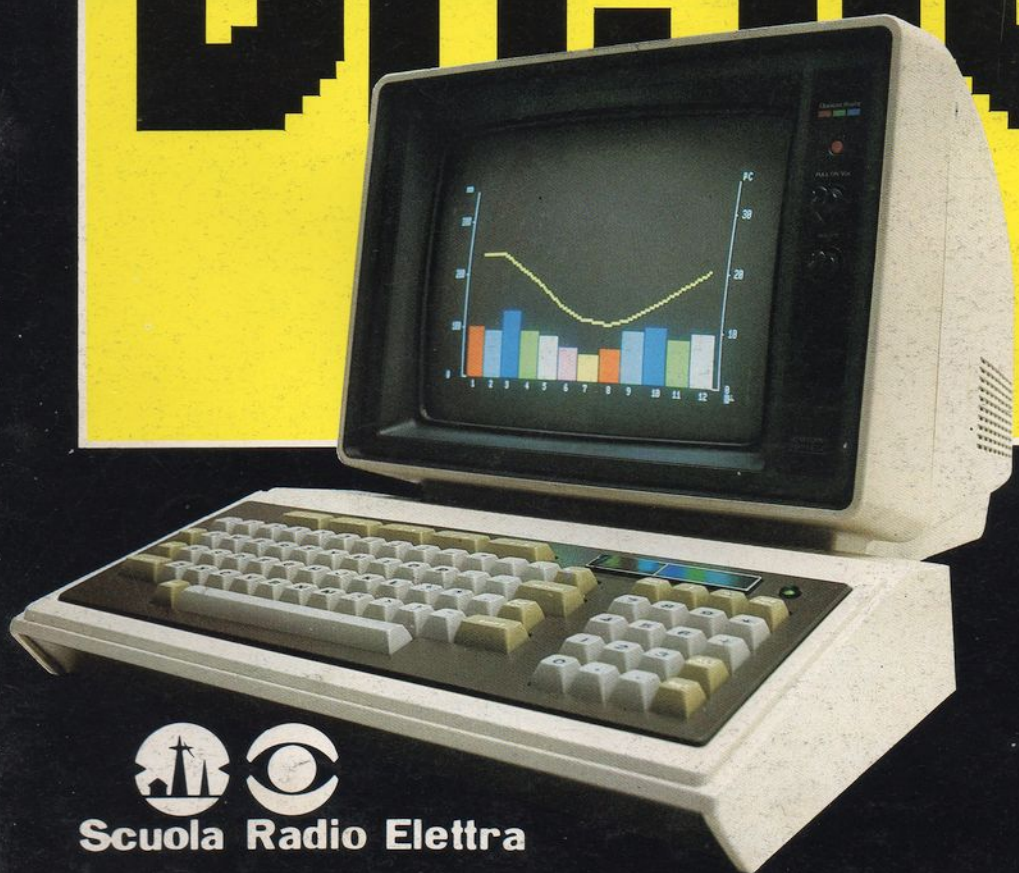
PARLA BASIC



Scuola Radio Elettra

Scuola Radio Elettra
Torino

Analisi e programmazione

Lezione 7

1 — Che cos'è il loop di un programma
2 — Gli enunciati **FOR... NEXT...**
3 — Come si utilizzano i dati interni
4 — L'istruzione **GET**
5 — Le istruzioni **POKE, PEEK, WAIT**

Che cosa apprenderà

In questa lezione verranno esaminate le istruzioni **FOR... NEXT...** che consentono di realizzare i cosiddetti cicli di programma, ossia i flussi ripetitivi di elaborazione.

Si vedrà quindi come tali istruzioni possano essere utilizzate piú volte nel programma per realizzare piú cicli di elaborazione, uno interno all'altro.

Verranno prese, inoltre, in considerazione le istruzioni **DATA**, **READ** e **RESTORE**, con le quali è possibile inserire nel programma liste di dati.

La lezione proseguirà con lo studio dell'istruzione **GET**, che viene normalmente utilizzata per trasferire il controllo dell'elaborazione all'operatore, ossia all'esterno del sistema.

Infine, verranno esaminate le istruzioni **POKE**, **PEEK**, ecc. che permettono di accedere direttamente alla memoria del computer.

1. - Che cos'è il loop di un programma

LOOP è una parola inglese che significa "CICLO", cioè ripetizione, ricalcolo di una espressione, ripristino o aggiornamento di certi valori, ecc.

Nella *fig. 1* è rappresentato, a titolo esemplificativo, il diagramma di flusso che realizza un ciclo di programma, il quale, nel caso specifico, calcola e stampa i quadrati dei primi dieci numeri interi.

In questa figura il blocco 1) del diagramma sta a rappresentare l'assegnazione del primo numero intero (uno) alla variabile di controllo di nome I; si inizializza cioè il ciclo ponendo I=1.

Il blocco 2) calcola il quadrato della variabile I ed assegna questo risultato alla variabile di nome A, che poi viene visualizzata, unitamente a I, nel blocco 3).

Nel blocco di diramazione 4) viene fatto un confronto tra il valore della variabile I ed il numero 1Ø (che è il massimo valore della variabile di controllo assunto nel ciclo).

Se dal confronto risulta che I è inferiore a 1Ø, il controllo del programma passa al blocco 5) e poi "ritorna" o "cicla", come si usa anche dire, al punto di ricalcolo 2).

Ogni volta che il ciclo passa al punto 5) la variabile di controllo I subisce un'aggiunta di un'unità (1), ossia viene aggiornata; quindi, se I parte dal valore 1 e deve arrivare al valore massimo 1Ø con l'incremento ogni volta di una unità, l'elaborazione "fluirà" o "ciclerà" per i punti o blocchi 2), 3), 4) e 5) esattamente dieci volte.

Quando I avrà raggiunto il valore 1Ø, il confronto raffigurato nel blocco 4) risulterà positivo ed il programma devierà per la via "si", cioè uscirà dal ciclo.

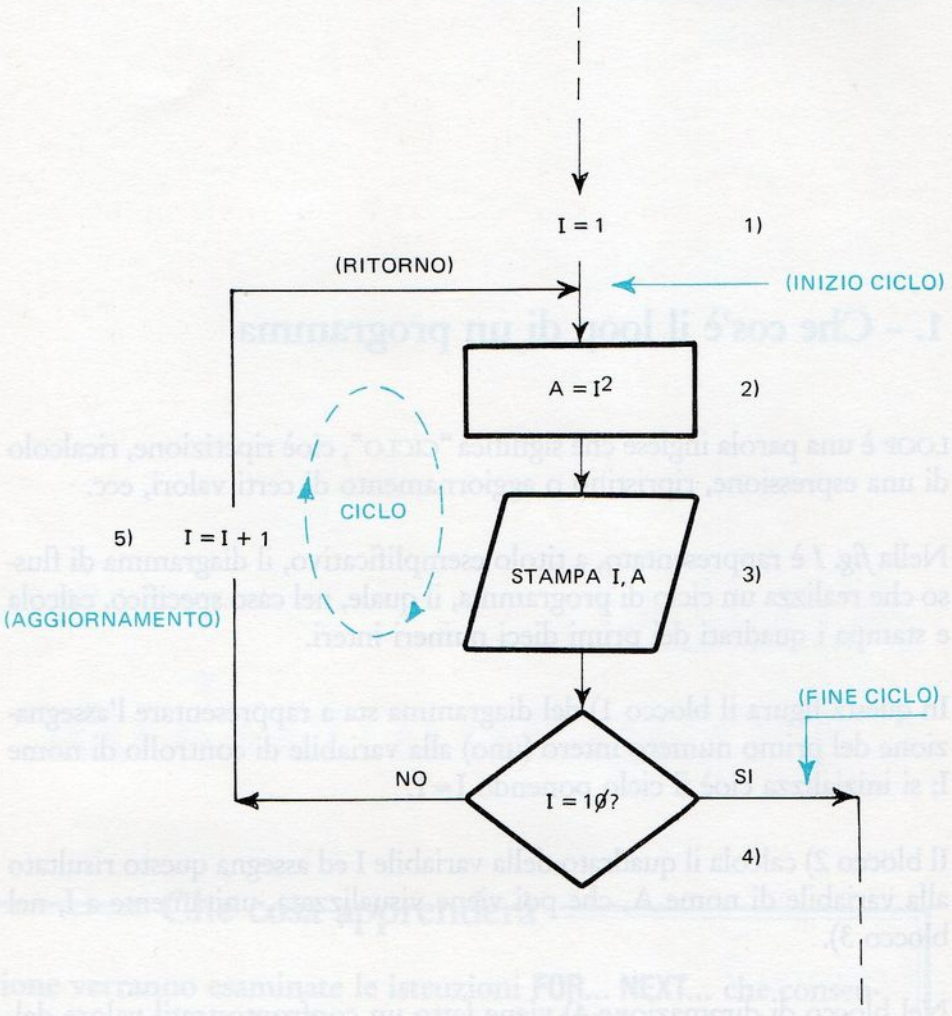


Fig. 1 – Ciclo di programma che calcola e stampa i quadrati dei primi dieci numeri interi.

Come si può rilevare da questo semplice esempio, un ciclo di programma è caratterizzato dalle seguenti proprietà.

a) Una variabile numerica, detta anche **variabile di controllo** o **indice contatore**, all'inizio del ciclo assume un valore intero; successivamente, ad ogni esecuzione del ciclo, la variabile aumenta di una certa quantità, detta anche **passo del ciclo**. Infine questa stessa variabile, arrivando ad assumere un altro valore finale, fa deviare l'elaborazione fuori del ciclo.

b) La parte di programma compresa in un ciclo viene elaborata tutte le volte che il controllo del programma ritorna all'inizio del ciclo, cioè ogni volta che la variabile di controllo assume un nuovo valore. In un

diagramma di flusso, l'inizio e la fine di un ciclo sono indicati con simboli grafici, come ha già potuto verificare in precedenza.

In particolare, la fine del ciclo avviene quando la variabile di controllo I assume il massimo valore 1Ø. Il simbolo di diramazione semplice (rombo contraddistinto con la scritta I=1Ø) controlla appunto se il valore I è arrivato al valore 1Ø.

c) Il valore numerico della variabile di controllo, dopo ogni passo di ciclo, e prima di ritornare all'inizio per eseguire il successivo passo, aumenta di una quantità detta **passo** o **incremento di elaborazione**. L'aggiornamento avviene in una parte "isolata" del programma dopo che il punto di controllo indicato con il simbolo di diramazione semplice ha dato esito negativo.

Dopo l'aggiornamento della variabile di controllo, l'elaborazione ritorna ad eseguire i passi di programma indicati dai punti 2) e 3) sempre nella *fig. 1* (si dice anche che l'elaborazione ritorna in ciclo).

Per programmare in BASIC un ciclo di programma possono essere usate diverse istruzioni, tra le quali l'istruzione condizionale **IF... THEN....**

Usando l'istruzione **IF... THEN...**, la variabile di controllo I deve essere preliminarmente inizializzata al valore di partenza, inoltre, la fine del ciclo di programma viene controllata dalla stessa istruzione

IF (I) **THEN** (A)

con la quale il controllo dell'elaborazione passa alla parte di programma indicata da (A) se (IF) il valore della variabile di controllo I risulta uguale ad un certo valore massimo (N); altrimenti prosegue ad "aggiornare" la variabile di controllo stessa e ritorna al punto di partenza del ciclo.

Un possibile listato del programma della *fig. 1*, relativo al calcolo dei quadrati dei primi dieci numeri interi, è riportato di seguito.

```
10 REM*QUADRATI DEI PRIMI 10 NUMERI*
20 I=1
30 A=I↑2
40 PRINT I, A
50 I=I+1
60 IF I<=10 THEN 30
70 END
```

Come si può rilevare, in questo caso è stata appunto utilizzata l'istruzione **IF... THEN...** per programmare il ciclo.

Inoltre, l'istruzione 4Ø provvede a visualizzare sia il valore della variabile A, che è il quadrato del numero intero I, sia l'intero I stesso nella prima posizione dello schermo.

Come si può constatare osservando il listato in questione, gli elementi fondamentali di un ciclo sono:

- inizializzazione della variabile di controllo (istruzione 2Ø);
- corpo del ciclo (istruzioni 3Ø e 4Ø);
- aggiornamento della variabile di controllo (istruzione 5Ø);
- confronto di uscita (istruzione 6Ø).

Un altro criterio per programmare un ciclo di programma, come ora si vedrà, è quello basato sull'uso dell'istruzione **FOR... NEXT...**, la quale è piú compatta della **IF... THEN...**, nel senso che non impiega istruzioni di salto incondizionato (**GOTO**), ed è maggiormente identificabile.

2. – Gli enunciati FOR... NEXT...

La parola chiave **FOR...**, seguita da alcuni parametri, sta ad indicare **l'inizio del ciclo di programma**, mentre la parola chiave **NEXT...** indica **la fine** dello stesso.

I parametri espressi dopo la parola FOR servono a delimitare gli estremi (inferiore e superiore) del ciclo di elaborazione: tali estremi sono i valori minimo e massimo che può avere la variabile di controllo.

Il loop realizzato con FOR... NEXT...

Nel suo formato piú generale, l'istruzione **FOR... NEXT...** si scrive nel seguente modo:

```
num. 1 FOR  X=N1 TO N2 STEP N3
        :
num. 2 NEXT X
        :
```

CICLO

In questa espressione num. 1 e num. 2 sono rispettivamente i numeri di riferimento delle istruzioni **FOR** e **NEXT**; X è il nome della variabile di controllo (o variabile indice), che può avere soltanto valori reali interi (se X avesse un valore decimale, questo verrebbe ridotto all'intero piú vicino); N 1 e N 2 sono il minimo ed il massimo valore della variabile di controllo X (ossia i valori iniziali e finali del conteggio) e anch'essi possono essere solamente valori interi.

La parola chiave **STEP** significa PASSO e N 3 è la lunghezza del passo, cioè l'incremento che si vuole attribuire alla variabile di controllo X ad ogni ciclo; N 3 può essere un qualsiasi numero reale intero.

Il valore del passo, ossia N 3, può anche essere negativo; in tal caso il valore finale (N 2) deve essere inferiore al valore iniziale (N 1).

Se non viene scritto **STEP** N 3, il sistema implicitamente considera il ciclo variabile con passo 1, ossia l'incremento di X è pari a 1 ad ogni ripetizione del ciclo stesso.

NEXT X, come si è precisato, è la parola chiave che, posta alla fine del ciclo, ne definisce la fine logica; X si riferisce alla variabile di controllo.

Quando il programma arriva ad un'istruzione **NEXT...**, ritorna all'istruzione **FOR...** e verifica che il valore X abbia raggiunto il valore massimo N 2. Se questo controllo è positivo, l'esecuzione del programma prosegue con l'istruzione successiva alla **NEXT...**.

Nel caso invece non si sia raggiunto il valore N 2, il programma riprende dall'istruzione successiva alla **FOR...**, incrementando la variabile X ad ogni ripetizione del ciclo.

Il valore iniziale N 1, il valore finale N 2 e la lunghezza del passo N 3 possono, a loro volta, essere delle espressioni.

Cosí l'istruzione **FOR...** potrebbe, ad esempio, essere:

```
10 FOR X=A+1 TO B*3 STEP D+C.
```

I valori A, B*3, D+C sono assegnati o calcolati prima.

Si può ora riscrivere il programma visto prima che calcola il quadrato dei

primi dieci numeri interi, utilizzando l'istruzione **FOR... NEXT...**; esso sarà così strutturato:

```
10 REM*QUADRATI DEI PRIMI 10 NUMERI*
20 FOR I=1 TO 10
30 A=I↑2
40 PRINT I, A
50 NEXT I
60 END
```

Come si può osservare, la scrittura del programma risulta semplificata. Esaminiamo ora le fasi operative di tale programma, il cui diagramma di flusso è riportato nella *fig. 2*.

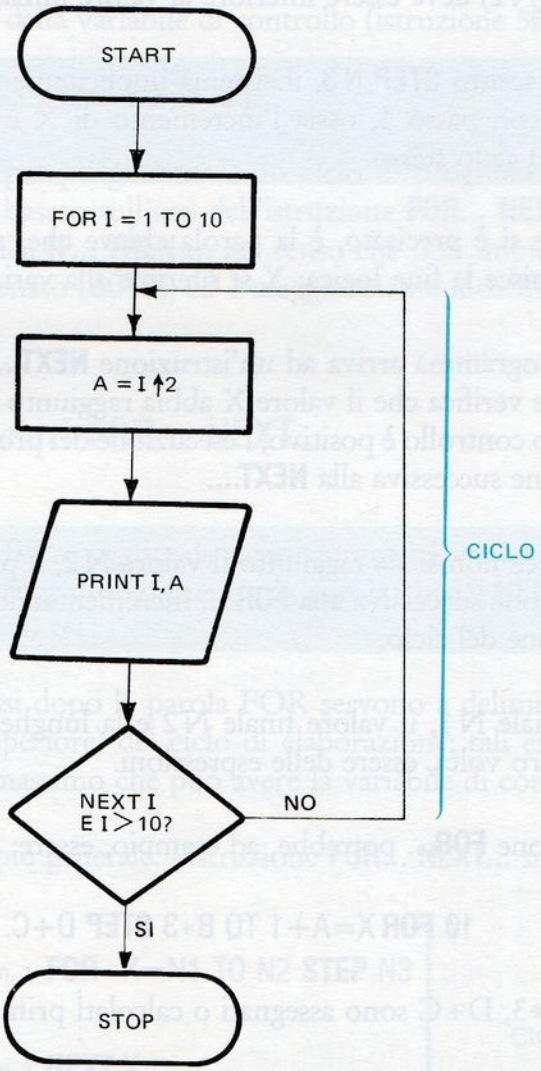


Fig. 2 – Flow-chart del programma che calcola il quadrato dei primi dieci numeri interi.

Per l'esecuzione del programma l'istruzione 2Ø impone per (FOR) I, il cui valore va da 1 a (TO) 1Ø, il calcolo e la stampa del quadrato di I (istruzioni 3Ø e 4Ø).

Quindi, l'istruzione 5Ø aumenta di 1 il valore di I (NEXT) ed il controllo ritorna all'istruzione 2Ø.

Il ciclo si ripete sino a quando I=1Ø; a questo punto il ciclo si ferma e viene eseguita l'istruzione 6Ø (**END**).

Riassumendo, l'istruzione 2Ø (**FOR**) inizializza e controlla l'uscita del ciclo, mentre l'istruzione 5Ø (**NEXT**) modifica la variabile di controllo I e rimanda all'istruzione 2Ø.

Le istruzioni 3Ø e 4Ø, poste fra le istruzioni **FOR...** e **NEXT...** che costituiscono il corpo del ciclo, vengono quindi ripetute dieci volte.

Volendo, ad esempio, calcolare i quadrati dei primi dieci numeri interi pari, si può modificare il programma ora visto in questo modo:

```
1Ø REM*QUADRATI DEI PRIMI 1Ø NUMERI PARI*
2Ø FOR I=2 TO 2Ø STEP 2
3Ø A=I↑2
4Ø PRINT I, A
5Ø NEXT I
6Ø END
```

La variabile I assume inizialmente il valore 2 e quindi vengono stampati i due numeri 2 e 4. Successivamente il ciclo viene ripetuto una seconda volta ed I assume il valore 2+2 (per cui vengono stampati i numeri 4 e 16); alla terza ripetizione del ciclo, I assume il valore 4+2=6 e cosí via, sino a quando, alla ripetizione del decimo ciclo, I=2Ø. Ad ogni ciclo la variabile I viene quindi incrementata di 2 (**STEP** 2).

Se invece si desidera calcolare i quadrati dei primi dieci numeri interi dispari, si deve scrivere quest'altro programma:

```
1Ø REM*QUADRATI DEI PRIMI 1Ø NUMERI DISPARI*
2Ø FOR I=1 TO 19 STEP 2
3Ø A=I↑2
4Ø PRINT I, A
5Ø NEXT I
6Ø END
```

Volendo, infine, calcolare i quadrati dei primi dieci numeri interi dispari e stamparli in ordine **decrescente**, si dovrà scrivere il programma nel modo seguente:

```
10 REM*QUADRATI DEI PRIMI 10 NUMERI
DISPARI*
20 FOR I=19 TO 1 STEP -2
30 A=I↑2
40 PRINT I, A
50 NEXT I
60 END
```

In questo caso il passo (STEP) è negativo (–2) ed il valore finale (1) della variabile di controllo I è inferiore al valore iniziale (19).

Una caratteristica peculiare che si può notare negli esempi fatti è che **la variabile di controllo non appare mai alla sinistra del segno uguale** di un'istruzione di assegnazione, cioè **non può subire alterazione di valore** all'interno di un ciclo.

Ciò è motivato dal fatto che la funzione di questa variabile è essenzialmente quella di controllare, ossia di contare il numero dei passi di elaborazione che devono essere effettuati in un ciclo.

2.1 – Caratteristiche dell'istruzione FOR... NEXT...

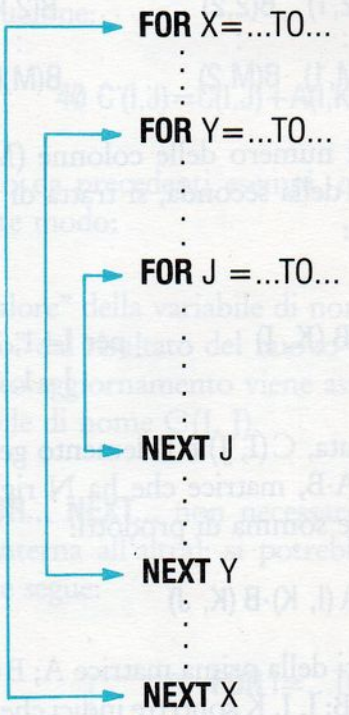
Cicli nidificati

E' possibile utilizzare l'istruzione **FOR... NEXT...** piú volte nel programma per realizzare piú cicli di elaborazione, uno interno all'altro.

La condizione che però occorre osservare è che questi cicli non siano incrociati e che le variabili di controllo dei vari cicli abbiano nomi diversi.

Lo schema logico di un gruppo di cicli interni, detti anche **cicli nidificati**

(nested loop, in inglese), è di seguito rappresentato:



Come si può rilevare, ogni ciclo interno è completamente contenuto nel ciclo esterno. Questo significa che l'istruzione **NEXT** inerente il ciclo interno deve comparire prima dell'istruzione **NEXT** relativa al ciclo esterno.

Esempi di utilizzazioni di cicli nidificati si verificano spesso in calcoli con le matrici, quali: prodotto di due matrici, calcolo del determinante di una matrice, ecc.

Esempi di cicli nidificati

Altri impieghi si hanno nell'elaborazione di strutture organizzate come i file (si legge "fail" e significa fila o insieme organizzato di informazioni).

Per il prodotto delle due seguenti matrici:

$$A(N,M) = \begin{pmatrix} A(1,1) & A(1,2) & \cdots & A(1,M) \\ A(2,1) & A(2,2) & \cdots & A(2,M) \\ \vdots & \vdots & & \vdots \\ A(N,1) & A(N,2) & \cdots & A(N,M) \end{pmatrix}$$

$$B(M,P)=\begin{pmatrix} B(1,1) & B(1,2) & \ldots\ldots & B(1,P) \\ B(2,1) & B(2,2) & \ldots\ldots & B(2,P) \\ \vdots & \vdots & & \vdots \\ B(M,1) & B(M,2) & \ldots\ldots & B(M,P) \end{pmatrix}$$

caratterizzate dal fatto che il numero delle colonne (M) della prima è uguale al numero delle righe della seconda, si tratta di programmare la seguente formula matematica:

$$1)\ C(I,J)=\sum_{K=1}^{M} A(I,K)\cdot B(K,J) \qquad \begin{matrix} \text{per } I=1,\ 2,\ldots.N \\ J=1,\ 2,\ldots.\ P \end{matrix}$$

Nell'espressione ora considerata, C(I, J) è l'elemento generico della matrice, risultato del prodotto A·B, matrice che ha N righe e P colonne; ogni elemento si ottiene come somma di prodotti:

$$A(I,K)\cdot B(K,J)$$

dove A(I, K) sono gli elementi della prima matrice A; B(K, J) sono gli elementi della seconda matrice B; I, J, K sono tre indici che variano rispettivamente da 1 a N (il primo), da 1 a P (il secondo), da 1 a M (il terzo).

E' chiaro che per calcolare gli N·P elementi della matrice prodotto C occorrono N·M·P somme di prodotti, date dalla formula 1) scritta sopra.

Per programmare in BASIC la stessa formula occorrono tre cicli o loop di programma, uno interno all'altro, come di seguito indicato:

```
10 FOR I= 1 TO N
20 FOR J=1 TO P
25 C(I,J)=0
30 FOR K=1 TO M
40 C(I,J)=C(I,J)+A(I,K)*B(K,J)
50 NEXT K
60 NEXT J
70 NEXT I
```

Come si può notare, l'istruzione 25 del programma in esame consente di azzerare gli elementi della matrice C(I, J) prodotto, prima che vengano calcolati dall'istruzione:

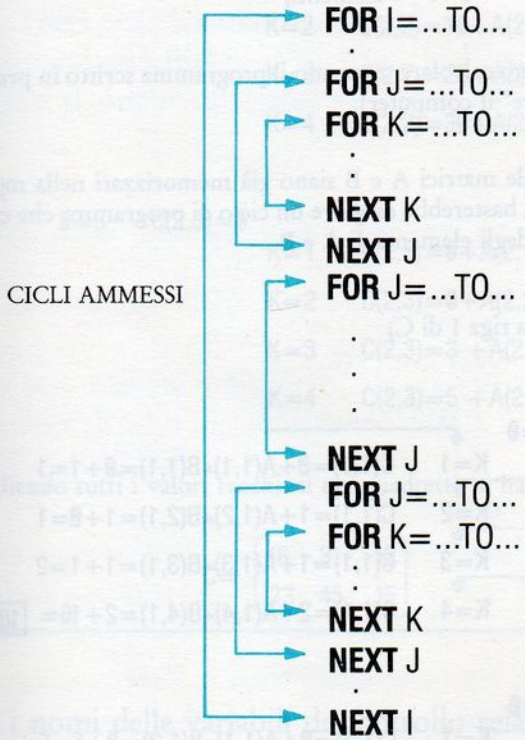
```
40 C(I,J)=C(I,J)+A(I,K)*B(K,J).
```

Come già è noto da precedenti esempi, questa istruzione di calcolo si legge nel seguente modo:

Il "precedente valore" della variabile di nome C(I, J) viene "incrementato" o "aggiornato" dal risultato del nuovo prodotto A(I,K)*B(K,J) ed il risultato di questo aggiornamento viene assegnato come "nuovo valore" alla stessa variabile di nome C(I, J).

Regole formali della FOR... NEXT...

Le istruzioni **FOR... NEXT...** non necessariamente devono essere nidificate (cioè una interna all'altra); si potrebbe verificare il caso descritto nello schema che segue:



Come si può notare, il ciclo piú grande con variabile di controllo I comprende diversi cicli "disgiunti", i quali, a loro volta, racchiudono altri cicli elementari.

Si può provare o "collaudare" il funzionamento logico del programma scritto precedentemente (a pag. 12) per calcolare il seguente prodotto di matrici:

$$C(2,3)=A(2,4)*B(4,3)$$

dove:

$$A(2,4)=\begin{pmatrix}1 & 2 & 1 & 4\\ 2 & 3 & 1 & 5\end{pmatrix}$$

$$B(4,3)=\begin{pmatrix}1 & 5 & 0\\ 0 & 6 & 1\\ 1 & 2 & 2\\ 4 & 3 & 2\end{pmatrix}$$

Si sa a priori che la matrice prodotto risultato C è costituita da due righe (tante quante sono le righe di A) e da tre colonne (tante quante sono le colonne di B), cioè da:

$$2\times 3=6 \text{ elementi,}$$

chiamati C (I, J), che si devono calcolare seguendo il programma scritto in precedenza (cioè cercando di "emulare" il computer).

Si supponga che i valori delle matrici A e B siano già memorizzati nella memoria principale; se non lo fossero, basterebbe eseguire un ciclo di programma che consenta un'immissione per righe degli elementi di A e B.

PASSO 1 DEL CICLO (calcolo della riga 1 di C)

```
I=1
      J=1    C(1,1)=0
                    K=1   C(1,1)=0+A(1,1)*B(1,1)=0+1=1
                    K=2   C(1,1)=1+A(1,2)*B(2,1)=1+0=1
                    K=3   C(1,1)=1+A(1,3)*B(3,1)=1+1=2
                    K=4   C(1,1)=2+A(1,4)*B(4,1)=2+16= [18]

I=1
      J=2    C(1,2)=0
                    K=1   C(1,2)=0+A(1,1)*B(1,2)=0+5=5
                    K=2   C(1,2)=5+A(1,2)*B(2,2)=5+12=17
                    K=3   C(1,2)=17+A(1,3)*B(3,2)=17+2=19
                    K=4   C(1,2)=19+A(1,4)*B(4,2)=19+12= [31]
```

I=1

J=3 C(1,3)=0

K=1 C(1,3)=0+A(1,1)*B(1,3)=0+0=0

K=2 C(1,3)=0+A(1,2)*B(2,3)=0+2=2

K=3 C(1,3)=2+A(1,3)*B(3,3)=2+2=4

K=4 C(1,3)=4+A(1,4)*B(4,3)=4+8= 12

PASSO 2 DEL CICLO (calcolo della riga 2 di C)

I=2

J=1 C(2,1)=0

K=1 C(2,1)=0+A(2,1)*B(1,1)=0+2=2

K=2 C(2,1)=2+A(2,2)*B(2,1)=2+0=2

K=3 C(2,1)=2+A(2,3)*B(3,1)=2+1=3

K=4 C(2,1)=3+A(2,4)*B(4,1)=3+20= 23

I=2

J=2 C(2,2)=0

K=1 C(2,2)=0+A(2,1)*B(1,2)=0+10=10

K=2 C(2,2)=10+A(2,2)*B(2,2)=10+18=28

K=3 C(2,2)=28+A(2,3)*B(3,2)=28+2=30

K=4 C(2,2)=30+A(2,4)*B(4,2)=30+15= 45

I=2

J=3 C(2,3)=0

K=1 C(2,3)=0+A(2,1)*B(1,3)=0+0=0

K=2 C(2,3)=0+A(2,2)*B(2,3)=0+3=3

K=3 C(2,3)=3 +A(2,3)*B(3,3)=3+2=5

K=4 C(2,3)=5 +A(2,4)*B(4,3)=5+10= 15

Raccogliendo tutti i valori racchiusi nei quadretti, si ha la matrice prodotto:

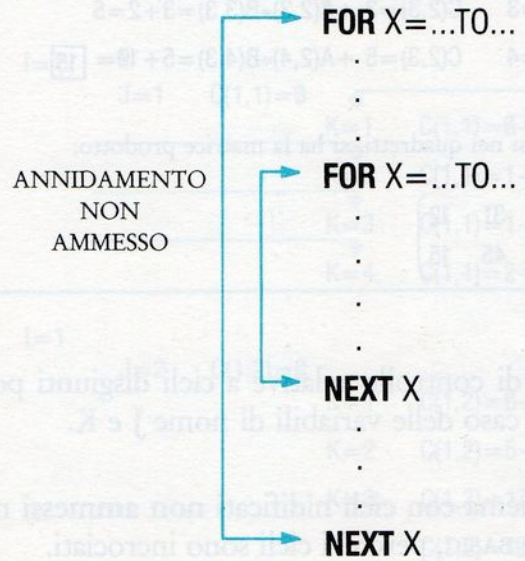
$$C = \begin{pmatrix} 18 & 31 & 12 \\ 23 & 45 & 15 \end{pmatrix}$$

Inoltre, i nomi delle variabili di controllo relative a cicli disgiunti possono essere ripetuti, come è il caso delle variabili di nome J e K.

Di seguito è riportato uno schema con cicli nidificati **non ammessi** nel linguaggio di programmazione BASIC, perché i cicli sono incrociati.

Infatti, come si può notare, il ciclo **FOR** J interno al ciclo **FOR** I non è nidificato in quest'ultimo, poiché la coda del ciclo **FOR** J (indicata dall'istruzione **NEXT** J) segue anziché precedere, come dovrebbe, la coda del ciclo **FOR** I.

Un altro annidamento non corretto è rappresentato dallo schema seguente:

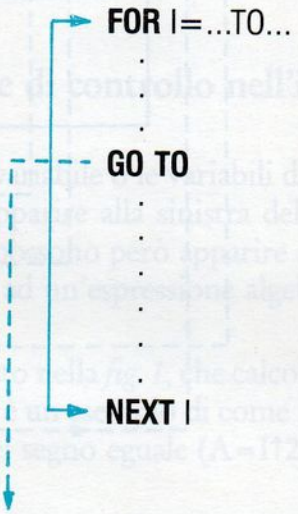


Come già accennato, questo annidamento è errato perché entrambi i cicli sono controllati dalla stessa variabile X.

Le istruzioni di salto **IF... THEN...** e **GOTO** possono essere usate nel ciclo **FOR... NEXT....**

Istruzioni di salto abbinate ad un ciclo

Un tipico esempio di tale possibilità è riportato nello schema seguente:

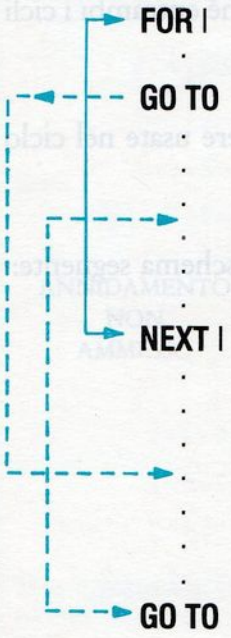


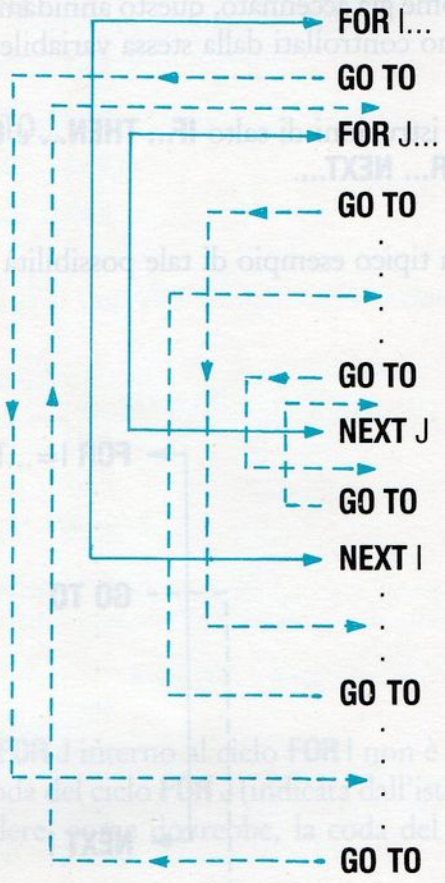
Come si può rilevare dall'esempio sopra raffigurato, nel ciclo **FOR I** c'è un'istruzione di salto incondizionato, **GOTO**, in un punto di programma "esterno" al ciclo **FOR I**.

Piú precisamente, dall'interno di un ciclo si salta in un punto esterno di programma e non si ritorna nello stesso ciclo **FOR I**.

In questo caso, dopo il salto **GOTO** fuori del ciclo **FOR I**, la variabile di controllo I conserva il medesimo valore che aveva al momento del salto **GOTO**.

Altri esempi di possibili situazioni ammesse sono raffigurati nella pagina seguente.





Quando tra la coda di un ciclo e un'altra non esistono istruzioni di programma, queste code possono essere unificate in una sola **NEXT**, seguita dal nome della variabile di controllo relativa al ciclo **FOR** piú interno.

Ad esempio, il programmino BASIC precedentemente considerato, che calcolava il prodotto di due matrici, avrebbe potuto essere ridotto, unificando le istruzioni di coda:

```
50 NEXT K
60 NEXT J
70 NEXT I
```

nell'unica coda:

```
50 NEXT K, J, I.
```

Inoltre, quando un ciclo non contiene altri cicli "annidati" internamente, si può omettere nella parola chiave **NEXT** l'unica variabile di controllo.

Per esempio, si può scrivere:

```
⋮
10 FOR I=1 TO 50
20 PRINT I
30 NEXT
⋮
```

cioè si può omettere il nome della variabile di controllo dopo la parola chiave **NEXT**, trattandosi di un ciclo che non contiene all'interno altri cicli.

2.2 - La variabile di controllo nell'istruzione FOR... NEXT...

La variabile di controllo della FOR... NEXT...

Come si è visto, la variabile o le variabili di controllo, se ci sono piú cicli, non devono mai apparire alla sinistra del segno uguale in istruzioni di assegnazione. Esse possono però apparire alla destra del segno uguale: ad esempio, in mezzo ad un'espressione algebrica.

Il programma esposto nella *fig. 1*, che calcola e stampa i quadrati dei primi dieci numeri interi, è un esempio di come la variabile di controllo (I) può apparire a destra del segno eguale (A=I↑2).

Per esempio, il programma che segue, il quale calcola i quadrati degli elementi di una matrice di numeri interi A (2,3) composta da due righe e tre colonne, è errato perché in esso la variabile di controllo delle colonne (J) appare alla sinistra del segno uguale di un'istruzione di assegnazione (linea di programma 40):

$$A(2,3)=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$$

```
10 REM*QUADRATI ELEMENTI MATRICE*
20 FOR I=1 TO 2
30 FOR J=1 TO 3
40 J=J↑2
50 A(I,J)=J
60 PRINT A(I,J)
70 NEXT J
80 NEXT I
```

In questo programma si è supposto che la matrice dei primi sei numeri interi (1, 2, 3, 4, 5, 6) sia già in memoria.

Per portarlo alla forma corretta si dovrebbero sostituire le istruzioni 4Ø, 5Ø e 6Ø con le seguenti due istruzioni:

```
:
40 Q=A(I,J)↑2
50 PRINT Q
:
```

oppure riscrivere il programma nel modo rappresentato nella *fig. 3*.

```
10 REM*******************
12 REM* QUADRATI DEI *
14 REM* PRIMI 6 NUME-*
16 REM* RI INTERI    *
18 REM*******************
20 DIM A%(2,3)
30 K%=1
40 FOR I=1 TO 2
50 FOR J=1 TO 3
60 A%(I,J)=K%↑2
65 PRINT K%,A%(I,J)
70 K%=K%+1
80 NEXT J
90 NEXT I
100 END
```

Fig. 3 - Quadrati dei primi sei numeri interi.

Quest'ultima codifica è piú completa della precedente (ed è corretta).

La matrice di interi di nome A% risulta costituita dai quadrati dei sei numeri interi 1, 2, 3, 4, 5, 6 che vengono generati dal programma stesso attraverso l'utilizzo della variabile intera K% (il cui valore iniziale è 1 e via via diventa 2, 3, 4, 5, 6).

3. - Come si utilizzano i dati interni

Nel linguaggio di programmazione BASIC sono contemplate particolari istruzioni, che hanno la funzione di immagazzinare dati numerici o stringa all'interno del programma stesso.

L'istruzione READ

Per utilizzare questi dati si scrive, sempre nel programma, l'istruzione BASIC di lettura **READ** (in inglese, leggere), seguita da nomi di variabili (numeriche o alfanumeriche) che corrispondono alle zone di memoria principale del computer in cui verranno depositati i dati.

La forma di scrittura corretta dell'istruzione in questione è quindi:

numero di riferimento **READ** elenco variabili

Ecco alcuni esempi di istruzioni **READ**:

```
10 READ A5, B, X%
20 READ BZ%, A$, C2$
```

L'istruzione DATA

L'istruzione BASIC avente la funzione di archivio interno di dati (che vengono poi letti con l'istruzione **READ**) è la **DATA**; essa è preceduta da un numero di riferimento, come se si trattasse di una normale istruzione BASIC, e seguita da un elenco piú o meno lungo di costanti numeriche o stringa (alfanumeriche), le quali vanno separate da una virgola.

Normalmente, se il dato è costituito da una stringa **non contenente all'interno spazi, virgole** o **due punti**, non occorre anteporre né posporre ad esso le virgolette.

Si devono invece usare le virgolette quando esistono all'interno della stringa spazi iniziali o finali, oppure caratteri speciali, come il punto (.), il punto e virgola (;), la virgola (,), i due punti (:).

La sintassi dell'istruzione **DATA** è la seguente:

numero di riferimento **DATA** elenco costanti

Esempi di istruzioni **DATA:**

```
100 DATA 3.14159, "SCUOLA RADIO ELETTRA", FRED
110 DATA CRISTIANO, 33E-5, AAA 123
```

Come si vede, i valori propriamente numerici (3.1159) vanno scritti con il solito punto decimale e senza le virgolette, come se si trattasse di una costante numerica, mentre le stringhe contenenti qualche spazio all'interno sono racchiuse tra virgolette e quelle prive di interspazi sono scritte senza virgolette.

Quando si ha un elenco piuttosto lungo di dati da immagazzinare in un programma, si può far uso di piú istruzioni DATA, ognuna con un proprio numero di riferimento. Il dato finale di ogni istruzione **DATA** è sempre privo della virgola.

Ogni volta che viene eseguita un'istruzione di lettura **READ** nel programma, il computer sposta il puntatore di posizione all'inizio del prossimo dato da leggere. In pratica, l'elenco di dati viene letto nella memoria principale, allo stesso modo in cui verrebbe letto un file di dati da un nastro magnetico.

Le istruzioni **DATA** possono essere collocate in qualsiasi punto del programma, però è buona regola inserirle in coda al programma stesso prima dell'istruzione **END**; in tal modo risultano piú facilmente rintracciabili e modificabili.

Come risulta evidente da quanto sinora precisato, le istruzioni **READ** e **DATA vengono sempre usate insieme.**

Il semplice programma che segue ha lo scopo di illustrare il funzionamento delle due istruzioni in questione.

```
10 READ A, B
20 READ C, D
30 PRINT A, B, C, D
40 DATA 1.2, 3.5
50 DATA 6, 7.8
60 END
```

In questo programma alle variabili A, B, C, D indicate nelle istruzioni **READ** (10 e 20) vengono assegnati rispettivamente i valori specificati nelle istruzioni **DATA** (40, 50).

Ossia, alla variabile A viene assegnato il valore 1.2, a B il valore 3.5, a C il valore 6 e a D il valore 7.8.

L'istruzione **PRINT** (30) provvede ovviamente a stampare sullo schermo i risultati dell'assegnazione, cioè i numeri:

```
 1.2        3.5          6          7.8
```

Non è indispensabile che in un programma il numero delle istruzioni **READ** sia uguale al numero delle istruzioni **DATA**. Ad esempio, nel pro-

gramma considerato le istruzioni 4Ø e 5Ø possono essere scritte in una sola istruzione:

```
4Ø DATA 1.2, 3.5, 6, 7.8
```

Una avvertenza da tenere, invece, sempre ben presente è che il numero dei valori riportati in una o piú istruzioni **DATA deve essere uguale complessivamente alle variabili specificate nelle istruzioni READ.**

Nel caso che nel programma il numero dei dati specificati nella istruzione **DATA** sia superiore al numero delle variabili elencate nell'istruzione **READ**, **i dati in sovrappiú non vengono considerati.**

Se invece i dati forniti sono inferiori al numero delle variabili, il computer emette il seguente messaggio d'errore:

```
? OUT OF DATA
```

ossia "dati insufficienti" e l'elaborazione termina.

A titolo esemplificativo, si consideri il seguente programma, che esegue il calcolo della circonferenza di quattro cerchi, aventi rispettivamente i raggi di 5.2, 4.5, 6.7, 8.

```
10 REM*CALCOLO CIRCONFERENZA DI 4 CERCHI*
15 READ R
20 DATA 5.2, 4.5, 6.7, 8
25 PRINT R*2*3.14159
30 GOTO 15
35 END
```

Quando viene eseguita l'istruzione 15, questa legge il primo valore di raggio (R), ossia il valore 5.2 specificato nell'istruzione **DATA** (2Ø).

Quindi, il programma provvede a calcolare e a stampare (istruzione 25) la lunghezza della circonferenza del cerchio di raggio 5.2, che risulta 32.672536.

A questo punto l'istruzione **GOTO** (3Ø) fa tornare il controllo del programma all'istruzione 15 ed alla variabile R viene assegnato il secondo valore del blocco dati, ossia il valore 4.5; viene cosí calcolata e stampata la lunghezza della circonferenza del cerchio di raggio 4.5, che è pari a 28.27431.

L'istruzione 3Ø fa poi nuovamente ritornare il controllo all'istruzione 15, e cosí via. Al termine del calcolo della quarta circonferenza (raggio 8), non essendoci altri dati disponibili nell'istruzione **DATA**, si ha la generazione del messaggio:

```
? OUT OF DATA
```

il quale indica che è stata tentata una lettura senza esito del quinto valore, e l'elaborazione si ferma.

Eseguendo il programma in questione con il comando **RUN**, sullo schermo compariranno i valori delle lunghezze dei quattro cerchi:

```
 32.672536
 28.27431
 42.097306
 50.26544
? OUT OF DATA
 ERROR IN 10
```

Se nel programma ora considerato fosse stato fornito un maggior numero di dati, cioè piú circonferenze da calcolare, esso avrebbe continuato ad elaborare fino al loro esaurimento per poi fermarsi automaticamente.

Come ulteriore esempio di utilizzazione delle istruzioni **READ** e **DATA**, si può considerare il seguente programma, che visualizza i numeri 1, 2, 3, immagazzinati in un'istruzione **DATA**.

```
10 REM*VISUALIZZA 1, 2, 3,*
20 FOR I=1 TO 3
30 READ A%
40 PRINT:PRINTA%:PRINT
50 NEXT I
60 DATA 1, 2, 3
70 END
```

Con questo programma i tre numeri 1, 2, 3 apparirebbero sullo schermo su tre linee diverse ed intercalati da un'interlinea (l'istruzione 4Ø provvede a questo tipo di visualizzazione).

Come si può notare, per visualizzare i tre numeri si è utilizzato un ciclo con la **FOR... NEXT...**: in ognuno dei tre passi del ciclo il dato verrà "letto" dall'elenco contenuto nell'istruzione **DATA**, immesso nel cassetto di me-

moria principale, chiamato A%, e successivamente visualizzato sullo schermo con l'istruzione 4Ø.

Dopo la prima lettura (o primo passo di ciclo, nel corso del quale la variabile di controllo I vale 1), il puntatore di posizione è posto sul secondo dato dell'elenco (che è il numero 2); dopo la seconda lettura esso è posto sul terzo dato (numero 3) e dopo la terza lettura resta posizionato dopo il numero 3.

L'istruzione **RESTORE**

Se si vuole leggere piú volte un elenco interno di dati (ricominciando sempre da capo), occorre eseguire un'istruzione appropriata chiamata:

RESTORE

la quale, se eseguita nel punto di programma in cui viene letto l'ultimo dato dell'elenco, provvede a riposizionare il puntatore all'inizio dell'elenco stesso.

Analizziamo ora l'effetto di tale istruzione, prendendo in considerazione il programma di seguito riportato:

```
1Ø READ A, B
15 RESTORE
2Ø READ C, D, E, F
25 PRINT A, B, C, D, E, F
3Ø DATA 1.1, 3.2, 4, 5.7
```

All'esecuzione dell'istruzione 1Ø vengono assegnate alle variabili A e B rispettivamente i valori 1.1 e 3.2. All'esecuzione dell'istruzione 15 (**RESTORE**) il puntatore di posizione dei DATA viene riportato dal secondo valore letto (3.2) al primo dato dell'elenco, per cui alle variabili C, D, E, F sono assegnati rispettivamente i valori 1.1, 3.2, 4, 5.7

Eseguendo questo programma con il comando **RUN**, sullo schermo appariranno quindi i seguenti valori:

```
 1.1   3.2   1.1   3.2   4   5.7
```

Nel programma riportato nella *fig. 4*, che calcola e visualizza la prima, la seconda e la terza potenza dei primi dieci numeri interi, sono presenti tutte le nuove istruzioni sinora studiate in questa lezione.

Per separare una serie di potenze dall'altra, cioè per raffigurarla in un'u-

```
10 REM*****************
12 REM*POTENZE PRIMI *
13 REM*  10 NUMERI   *
15 REM*    INTERI    *
20 REM*****************
30 FOR P=1 TO 3
35 PRINT:PRINT"POTENZA";P:PRINT
40 FOR I=1 TO 10
45 READ N$,D%
50 PRINT N$,D%↑P
60 NEXT I
65 RESTORE
67 STOP
70 NEXT P
80 DATA"NUMERO 1",1,"NUMERO 2",2,"NUMERO 3",3
85 DATA"NUMERO 4",4,"NUMERO 5",5,"NUMERO 6",6
90 DATA "NUMERO 7",7,"NUMERO 8",8,"NUMERO 9",9
95 DATA "NUMERO 10",10
100 END

READY.
```

Fig. 4 – Codifica del programma che calcola le prime tre potenze dei primi dieci numeri interi.

nica videata dello schermo, si potrebbe inserire dopo l'istruzione 30 una istruzione, numerata 33, che cancelli lo schermo e posizioni il cursore nel suo angolo superiore sinistro.

Alcuni computer usano per questo scopo l'istruzione BASIC **HOME**; altri, come il C64 ed il VIC-20 Commodore, utilizzano l'apposito tasto **CLR | HOME** .

Per i possessori del C64 e VIC-20 si può quindi scrivere:

(33 **PRINT** " **SHIFT** → **CLR | HOME** " **RETURN**)
33 **PRINT** " ♡ "

Si rammenta che l'istruzione racchiusa tra parentesi è quella che si deve praticamente battere per vedere comparire sullo schermo l'istruzione sottostante cosí com'è.

Il simbolo **SHIFT** → **CLR | HOME** sta a significare che prima bisogna premere il tasto **SHIFT** e poi, **tenendo quest'ultimo abbassato**, premere il tasto con il simbolo **CLR | HOME** .

Per eseguire il programma in questione, basta battere il comando BASIC **RUN**.

Si dovrebbero vedere tre prospetti (o videate), ognuno riportante la prima, la seconda, la terza potenza dei primi 1Ø numeri interi.

L'istruzione 65 **RESTORE** provvede a riportare il puntatore di posizione all'inizio dei dati interni, che sono stati memorizzati nelle due istruzioni comprese fra 8Ø e 95.

Ogni elemento informativo o dato è costituito da una stringa e da un valore numerico intero. La stringa viene assegnata, ad ogni istruzione **READ** (45), al nome stringa della memoria principale N$ e il numero intero al nome D%.

Si noti che l'ultimo dato di ogni istruzione **DATA**... finisce senza virgola.

Per dare modo all'operatore di controllare i risultati sul video, si è inserita l'istruzione di arresto temporaneo:

67 **STOP**

Quando l'elaborazione si arresta al numero di istruzione 67, l'operatore ha di fronte a sé uno dei tre prospetti di potenze che può cosí controllare.

Per continuare l'elaborazione occorre battere ogni volta il comando BASIC **CONT**.

4. – L'istruzione GET

L'istruzione **GET**, seguita da un nome di variabile, consente l'immissione, nella memoria principale del computer (nella locazione indicata dal nome della variabile), di un qualsiasi carattere numerico o stringa (uno solo) senza dover premere anche il tasto di consenso **RETURN**, o, per altri sistemi, il tasto **CR** oppure **ENTER** o tasti analoghi.

In pratica, l'istruzione **GET** è simile all'istruzione **INPUT**, ma con essa non c'è piú bisogno di premere il tasto di consenso ed **il carattere battuto non viene visualizzato sullo schermo.**

Inoltre, con l'istruzione **GET** un solo carattere per volta viene immesso nella memoria.

Per esempio, l'istruzione:

```
10 GET A$
```

consente di immettere da tastiera un qualsiasi carattere di quest'ultima (carattere che può essere letterale, grafico, numerico, ecc.).

Se si vuole visualizzare tale carattere sullo schermo, è necessario eseguire una successiva istruzione **PRINT**; ad esempio:

```
10 GET A$
20 IF A$="" THEN 10
30 PRINT A$
```

Il controllo di nessun carattere introdotto (istruzione 20 dell'esempio) deve essere effettuato, poiché il computer controlla ciclicamente il contenuto di un buffer di tastiera (area di memoria o registro), dove sono memorizzati i caratteri introdotti, indipendentemente dalle operazioni effettuate dall'operatore, in attesa di essere elaborati.

L'istruzione **INPUT**, invece, viene controllata, quando viene effettuata l'introduzione del carattere, mediante il tasto **RETURN**.

L'istruzione:

```
10 GET #1, A$
```

consente l'immissione, nella memoria principale del computer, di un carattere di un file di dati, registrato su una memoria ausiliaria (nastro magnetico, disco, ecc.), file a cui è stato associato il numero 1 (detto anche number-file).

Normalmente l'istruzione GET arresta l'esecuzione del programma e la fa riprendere quando viene battuto un tasto.

Utilizzazione della GET

L'istruzione **GET** viene utilizzata normalmente per "indagare" su una condizione esterna al sistema di elaborazione.

A volte l'elaborazione di un programma, per poter proseguire oltre, ha bisogno del consenso di un operatore esterno o, meglio, fa decidere a quest'ultimo circa la convenienza o meno di scegliere una via di programma anziché un'altra.

Con l'istruzione **GET** il controllo dell'elaborazione passa cioè all'operatore, il quale, in base al particolare valore della sua risposta, può decidere di far svolgere al programma un tipo di calcolo anziché un altro.

La risposta dell'operatore (costituita da un carattere qualsiasi della tastiera) viene esaminata successivamente all'istruzione **GET** con un'altra istruzione condizionale **IF...**, con la quale l'elaborazione proseguirà per una via o per un'altra in base al valore della risposta.

Per meglio chiarire la funzione dell'istruzione **GET**, riprendiamo il programma riportato nella *fig. 4* e supponiamo di modificarlo in modo tale che la prosecuzione del calcolo delle potenze sia decisa all'esterno del sistema di elaborazione, ossia che il controllo del programma sia affidato all'operatore.

Il programma, cioè, può iniziare a calcolare la potenza prima dei primi dieci numeri interi; poi chiede all'operatore (attraverso un'istruzione **GET**) se può proseguire a calcolare la successiva potenza, ossia la seconda, sempre dei primi dieci numeri interi.

Se l'operatore risponde ad esempio con il carattere S (che sta per SI), l'elaborazione continua e calcola la potenza seconda dei primi dieci numeri; se invece risponde con un qualsiasi altro carattere diverso da S, l'elaborazione si arresta.

Ad ogni esposizione di risultati sullo schermo, l'operatore osserverà i valori numerici delle potenze e deciderà conseguentemente se proseguire il calcolo delle stesse od arrestare l'elaborazione.

Naturalmente non potrà rispondere SI all'infinito, perché prima o poi il programma si arresterà da solo: esattamente quando i valori delle potenze supereranno il valore massimo (10^{38}) consentito nei sistemi personal computer.

La nuova codifica del programma che calcola le successive potenze dei primi dieci numeri interi è riportata nella *fig. 5*.

Come si può rilevare, il programma in questione è piú semplice del precedente, in cui per la creazione dei numeri interi erano state utilizzate istruzioni **DATA**. In esso, invece, le basi delle potenze vengono create dal programma stesso in un ciclo FOR (istruzioni 4Ø e 6Ø), utilizzando i valori correnti della variabile di controllo I, a cui, come piú volte si è avuto occasione di ribadire, non può essere assegnato alcun valore, ma

```
1 REM*****************
2 REM****POTENZE*****
3 REM*SUCCESSIVE DEI*
4 REM*    PRIMI 10   *
5 REM* NUMERI INTERI*
6 REM*****************
10 P=0
20 P=P+1
35 PRINT:PRINT"POTENZA";P:PRINT
40 FOR I=1 TO 10
50 PRINT"NUMERO=";I
55 PRINT"POTENZA=";I↑P
60 NEXT I
70 GET R$
80 IF R$="" THEN 70
85 IF R$="S" THEN 20
90 END
```

Fig. 5 - Codifica del programma che calcola le successive potenze dei primi dieci numeri interi.

può essere usata come argomento di un calcolo algebrico.

Con le istruzioni:

```
70 GET R$
80 IF R$="" THEN 70
```

viene chiesto all'operatore se vuole continuare l'elaborazione od arrestarla (si noti che le virgolette sono da battere una di seguito all'altra senza spazi interni).

Con tali istruzioni il computer "attende" che l'operatore batta da tastiera un carattere: se l'operatore batte un qualsiasi carattere diverso dal carattere S, l'elaborazione si arresta. Se invece batte S, l'elaborazione va avanti, finché non si raggiunge l'esponente P=38; a questo punto si arresta con il messaggio:

```
OVERFLOW
ERROR IN 55
```

ossia "supero di capacità".

Per eseguire il programma in questione occorre, come di consueto, scrivere il comando BASIC **RUN**.

Sullo schermo apparirà il primo prospetto di potenze, come di seguito riportato:

```
POTENZA    1

NUMERO  =1
POTENZA =1
NUMERO  =2
POTENZA =2
   ⋮
NUMERO  =10
POTENZA =10
```

Ad ogni battuta del tasto S compariranno sullo schermo i successivi prospetti delle potenze (ossia le potenze seconda, terza, quarta, ecc.).

Per arrestare l'elaborazione è sufficiente battere un qualsiasi altro carattere.

In alcuni tipi di computer, ad esempio nei Commodore, l'istruzione GET non fa visualizzare il cursore sullo schermo.

Come si vedrà in seguito, quando verrà trattata la gestione degli archivi nella memoria ausiliaria di massa (dischi e nastri), l'istruzione **GET** potrà essere utilizzata per immettere nella memoria principale un carattere per volta di un campo o record di informazioni.

5. – Le istruzioni POKE, PEEK, WAIT

L'enunciato **POKE** serve ad **introdurre** (cioè a scrivere) **particolari valori numerici interi** in una qualsiasi zona della memoria principale del computer (memoria RAM).

L'istruzione POKE

La parola chiave POKE è sempre seguita da due numeri: il primo rappresenta **un indirizzo di memoria**, il secondo, separato dal primo da una virgola, è costituito da un valore numerico intero compreso fra 0 e 255 e rappresenta il **byte da scrivere**.

A seconda dei particolari valori di questi due numeri, l'istruzione **POKE** può produrre dei suoni, raffigurare colori, creare programmi, spostare il cursore, configurare il monitor in svariati modi, ecc.

I valori numerici del primo numero o indirizzo possono essere compresi tra Ø e 65535.

In alcuni sistemi, come il VIC-20 per esempio, se si scrive:

POKE 36875,175

indirizzo — valore

viene eseguita una nota musicale (SOL), udibile attraverso l'altoparlante del sistema; se invece si scrive:

POKE 36879,27

viene raffigurata la combinazione colori con codice 27 (corrispondente al margine blu-verde e schermo bianco).

La funzione PEEK

Normalmente l'istruzione **POKE** viene utilizzata insieme alla funzione **PEEK** (sbirciare, in inglese), che **agisce da controllo per la prima**.

La funzione **PEEK** (X), infatti, consente di indagare, controllare il contenuto della locazione di memoria indicata da X.

Ad esempio, la funzione:

PEEK (78Ø)

permette di leggere il contenuto della locazione di memoria il cui indirizzo è 780 in decimale.

La funzione **PEEK** è sempre abbinata ad istruzioni **PRINT** o di assegnazione di variabili, come indicato negli esempi che seguono.

PRINT PEEK (78Ø)

visualizza sul video il contenuto della locazione 78Ø.

B=**PEEK** (76Ø)

memorizza nella variabile B il contenuto della locazione 76Ø.

Per interpretare correttamente i dati ricavati con la funzione **PEEK**, è indispensabile conoscere perfettamente la struttura del computer usato.

Occorre, inoltre, tenere ben presente che l'istruzione **POKE** deve essere utilizzata con la massima attenzione e prudenza, attenendosi sempre a quanto indicato nel manuale d'uso del computer.

Un impiego non corretto dell'istruzione **POKE** può, infatti, bloccare il funzionamento del computer stesso o causare errori nel programma od alterare eventuali dati registrati su disco.

L'istruzione WAIT

Esaminiamo ora l'istruzione **WAIT**; essa **consente l'inserimento di una pausa condizionale in un programma.**

Questa istruzione è sempre seguita da tre numeri interi: il primo corrisponde ad un indirizzo di memoria in cui è memorizzato un valore binario da "esaminare" per effettuare la pausa condizionale; gli altri due numeri (il secondo dei quali è opzionale) servono ad indicare per quali bit del primo numero binario va effettuata la pausa condizionale.

La forma corretta di questa istruzione è quindi:

WAIT LOCAZIONE, valore 1, valore 2
↑
opzionale

Gli ultimi due valori (numerici) sono compresi fra Ø e 255.

L'interruzione operata dall'istruzione **WAIT** dura sino a quando non risulta verificata, ossia vera, una relazione logica fra il contenuto della locazione specificata ed i due valori forniti.

L'istruzione PAUSE

Una istruzione analoga a quella ora esaminata è l'istruzione **PAUSE**, la quale consente di fermare l'esecuzione del programma per un certo periodo di tempo.

A differenza però della **WAIT**, nel caso della **PAUSE** l'esecuzione del programma riprende immediatamente premendo un tasto qualsiasi del computer.

A conclusione dell'argomento occorre precisare che è anche possibile creare una pausa durante l'elaborazione di un programma, usando un

ciclo **FOR... NEXT...** a vuoto e scegliendo convenientemente il numero dei cicli da eseguire; ad esempio:

```
:
50 FOR X=1 TO 1000:NEXT X
:
```

Allorché nel programma viene eseguita tale istruzione, l'elaborazione si ferma sino a quando X assume il valore 1000; dopo di ciò il programma riprende automaticamente, senza alcun intervento esterno.

* * * * * * *

In seguito...

Nella prossima lezione sarà esaminata l'organizzazione dei dati nella memoria principale del computer e nella memoria ausiliaria (dischi e nastri magnetici); saranno inoltre forniti cenni sull'organizzazione delle informazioni nei grandi sistemi di elaborazione.

Da memorizzare

- Il loop è una parola inglese che significa **ciclo**, cioè ripetizione, ricalcolo di una espressione, ripristino o aggiornamento di certi valori, ecc.

- La parola chiave FOR, seguita da alcuni parametri, sta ad indicare l'**inizio del ciclo** di programma, mentre la parola chiave NEXT indica la **fine** dello stesso.

- La forma generale dell'istruzione **FOR...** è: numero di riferimento, parola chiave FOR, variabile di controllo, segno uguale, numero o variabile (valore iniziale), parola chiave TO, numero o variabile (valore finale), parola chiave STEP (passo) seguita da un numero o una variabile.

- Quando non viene scritta la parola chiave STEP, il sistema implicitamente considera il ciclo variabile con passo 1.

- Per avere cicli di elaborazione uno interno all'altro (nidificati) occorre che essi **non siano incrociati** e che le variabili di controllo dei vari cicli **abbiano nomi diversi**.

- L'istruzione **GET...**, seguita da un nome di variabile, consente l'**immissione** nella memoria principale del computer **di un qualsiasi carattere numerico o stringa** (uno solo), senza dover premere il tasto **RETURN**. L'istruzione **GET** è simile all'istruzione **INPUT**.

- L'istruzione **DATA** consente di **immagazzinare** all'interno del programma **dati numerici o stringa**, che vengono poi letti con l'istruzione **READ**.

- Con l'istruzione **RESTORE** si può leggere piú volte un elenco interno di dati (ricominciando sempre dall'inizio).

- Normalmente l'istruzione **GET** fa arrestare l'esecuzione del programma e la fa riprendere quando viene battuto un tasto.

- L'istruzione **POKE** serve ad introdurre, ossia a scrivere, **particolari valori numerici interi** in una qualsiasi zona della memoria principale del computer.

- La funzione **PEEK** consente di **indagare**, **controllare** il contenuto della locazione di memoria del computer. Essa viene normalmente usata con l'istruzione **POKE**.

- L'istruzione **WAIT** permette l'inserimento di **una pausa condizionale** in un programma.

Scuola Radio Elettra
Torino

Pratica del BASIC

Lezione 7

1 — L'istruzione **FOR... NEXT...**
2 — Le istruzioni **READ** e **DATA**
3 — Programmazione a colori sul C64
4 — Programmazione a colori sul VIC-20

Che cosa farà

Nella prima parte di questa lezione verificherà praticamente l'uso dell'istruzione **FOR... NEXT...**, che costituisce il sistema piú efficiente per ottenere cicli di programmazione.

Inoltre, avrà modo di sperimentare l'uso delle istruzioni **READ** e **DATA**, che consentono la costruzione di tabelle di dati, eseguendo un programma di utilità di tipo fiscale.

Nella seconda parte della lezione si analizzeranno le possibilità da parte del computer di produrre immagini a colori e di simulare effetti di movimento utilizzando l'istruzione **POKE**.

1. – L'istruzione **FOR... NEXT...**

Come già precisato, il BASIC consente di semplificare la scrittura dei cicli di programma, ossia dei flussi ripetitivi di elaborazione, mediante l'uso dell'istruzione **FOR... NEXT...**, la quale risulta piú compatta dell'istruzione **IF... THEN...**, in quanto non richiede l'uso dell'istruzione di salto condizionato **GOTO**.

Vediamo ora di verificare sul computer i risvolti applicativi di questa nuova istruzione.

Esercizio 1

Come esempio di utilizzazione dell'istruzione **FOR... NEXT...** possiamo sviluppare il programma descritto nella lezione teorica del presente fascicolo, programma che calcola il prodotto di due matrici.

Affinché la moltiplicazione di due matrici si possa eseguire, è necessario che il numero delle colonne della prima matrice sia eguale al numero delle righe della seconda.

Gli elementi costituenti ciascuna matrice verranno introdotti nel computer mediante la tastiera.

Nella *fig. 1* è rappresentato il flow-chart del programma in questione, nel quale, come si può rilevare, sono presenti tre annidamenti di cicli: due costituiti da due cicli ed uno da tre cicli.

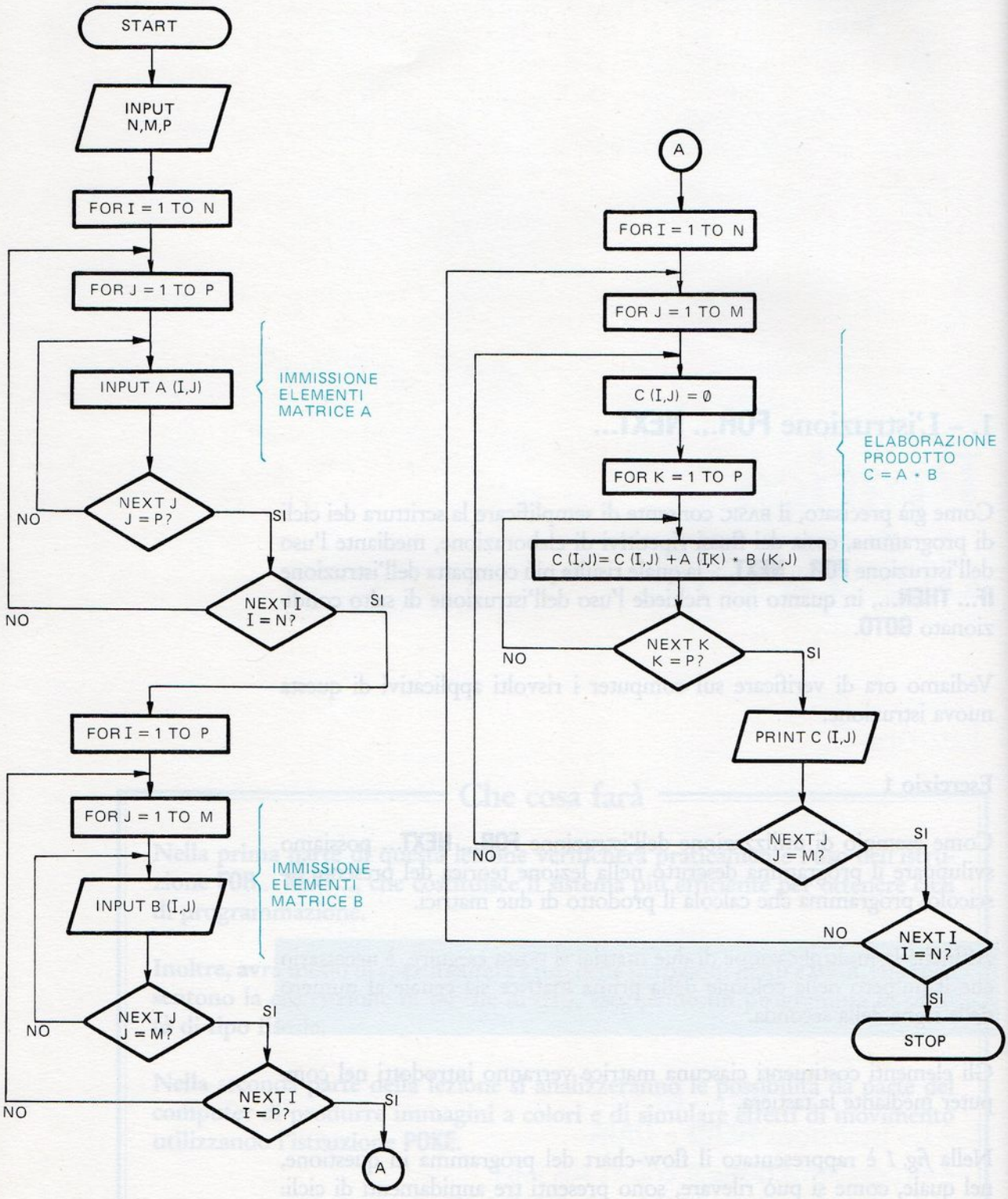


Fig. 1 – Flow-chart del programma che calcola il prodotto di due matrici.

In questo programma la costante N rappresenta il numero di righe della matrice A (che si introduce per prima nel computer), la costante M il numero di colonne della matrice B (che si introduce per seconda), la costante P il numero delle colonne di A oppure il numero delle righe di B.

La matrice risultato C avrà dimensioni N*M e il risultato sarà stampato o visualizzato alla fine dell'elaborazione un elemento per volta sulle linee dello schermo.

Osservando il diagramma di flusso della *fig. 1*, si può notare che esso, per comodità di rappresentazione, è stato "spezzato" in due parti ed il **simbolo di connessione**, costituito da un cerchietto nel quale è contenuta una lettera dell'alfabeto (nel nostro caso, la lettera A), indica appunto i due punti del diagramma che devono essere considerati direttamente collegati fra loro.

A titolo esemplificativo nella *fig. 2* è riportato un altro flow-chart del programma che calcola il prodotto di due matrici, il quale si differenzia dal precedente perché piú generalizzato.

Infatti, in questo nuovo diagramma di flusso i cicli annidati, detti anche nidificati, sono rappresentati senza specificare che sono realizzati con una istruzione **FOR... NEXT....**

Nella *fig. 3* è riportata la codifica, in linguaggio BASIC, del programma che calcola il prodotto di due matrici.

Esaminiamo ora le fasi operative piú significative di tale programma.

L'elaborazione del programma inizia con l'istruzione 25 di **INPUT**, nella quale compaiono le costanti N, M, P di cui è già stato precedentemente precisato il significato.

Nell'istruzione 3Ø vengono dimensionate le matrici A e B, che devono essere moltiplicate fra loro, nonché la matrice risultante C.

Gli elementi della matrice A, introdotti nel computer con l'istruzione 45 di **INPUT**, sono definiti da due cicli annidati: quello esterno, costituito dalle istruzioni 35 e 55 (numero delle righe) e quello interno, dalle istruzioni 4Ø e 5Ø (numero delle colonne).

Analogamente, gli elementi della matrice B, introdotti con l'istruzione 7Ø di **INPUT**, sono definiti mediante altri due cicli annidati, costituiti

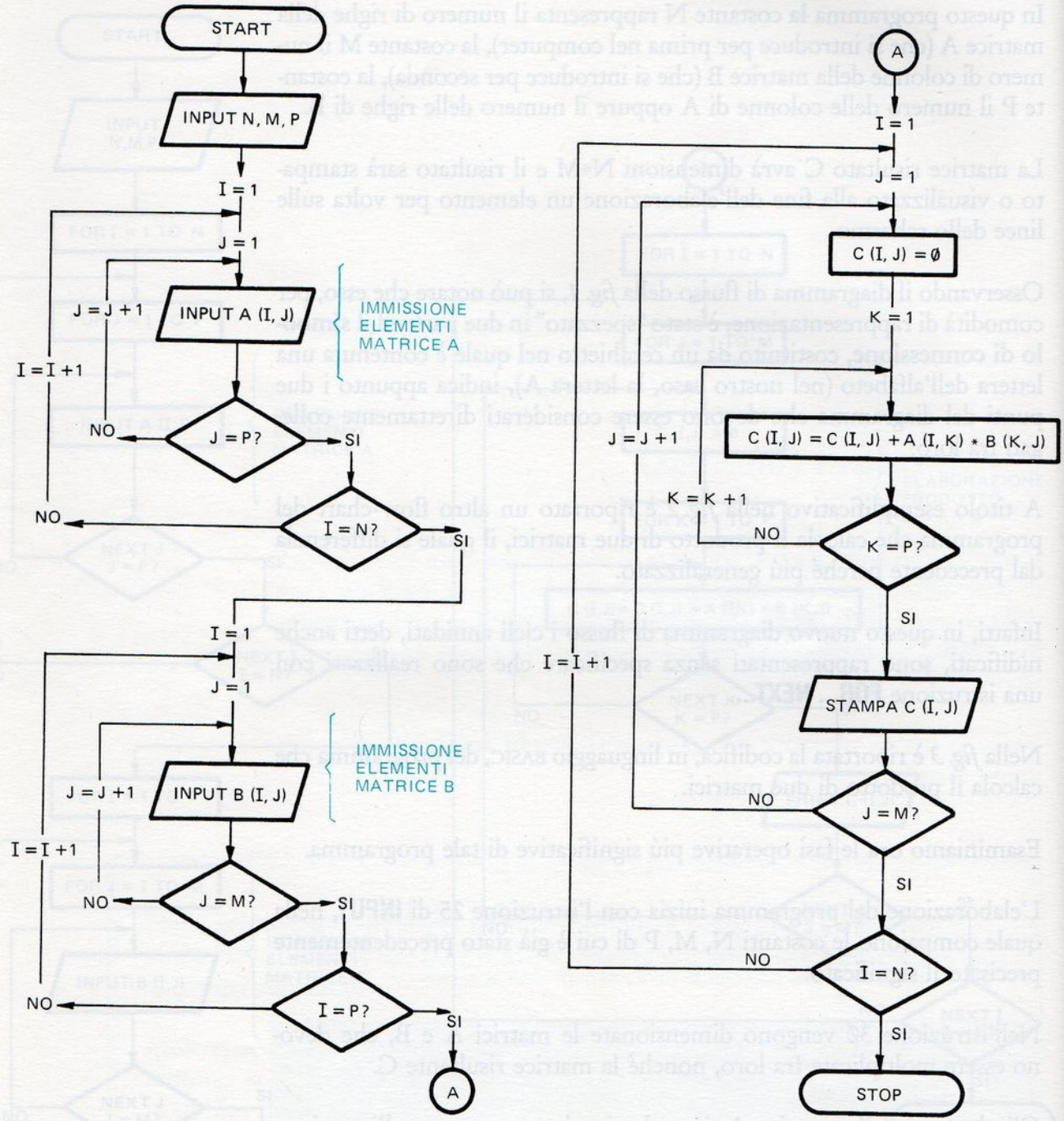


Fig. 2 – Altro flow-chart del programma che calcola il prodotto di due matrici.

rispettivamente dalle istruzioni 6Ø e 8Ø (numero delle righe) e dalle istruzioni 65 e 75 (numero delle colonne).

Il prodotto delle due matrici A e B viene ottenuto mediante un terzo

```
10 REM**************************
15 REM*PRODOTTO DI DUE MATRICI*
20 REM**************************
25 INPUT"INTR.N,M,P";N,M,P
30 DIM A(N,P),B(P,M),C(N,M)
35 FOR I=1 TO N
40 FOR J=1 TO P
45 INPUT"INTR.ELEM.A";A(I,J)
50 NEXT J
55 NEXT I
60 FOR I=1 TO P
65 FOR J=1 TO M
70 INPUT"INTR.ELEM.B";B(I,J)
75 NEXT J
80 NEXT I
85 FOR I=1 TO N
90 FOR J=1 TO M
95 C(I,J)=0
100 FOR K=1 TO P
105 C(I,J)=C(I,J)+A(I,K)*B(K,J)
110 NEXT K
115 PRINT"----------------------"
120 PRINT"C(";I;",";J;")";"=";C(I,J)
125 NEXT J
130 NEXT I
135 END
```

Fig. 3 – Codifica del programma che calcola il prodotto di due matrici.

annidamento, costituito da tre cicli: il primo, formato dalle istruzioni 1ØØ e 11Ø, è annidato nel ciclo definito dalle istruzioni 9Ø e 125, che a sua volta è annidato nel ciclo costituito dalle istruzioni 85 e 13Ø. In particolare, l'istruzione 1Ø5, inserita nel ciclo piú interno del terzo annidamento, consente di calcolare i valori degli elementi della matrice risultante C. Questi valori vengono poi stampati sullo schermo mediante l'istruzione 12Ø.

Esaminato il programma, può ora immetterlo nel computer battendo con cura le istruzioni che lo costituiscono.

Per "collaudare" questo programma, proveremo ad introdurre nel computer le due matrici seguenti:

$$A(2,3)=\begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$B(3,2)=\begin{pmatrix} -1 & 0 \\ 1 & 1 \\ 3 & 2 \end{pmatrix}$$

Il prodotto A(2,3)*B(3,2) dovrebbe risultare:

$$C(2,2)=\begin{pmatrix} 4 & 4 \\ 4 & 3 \end{pmatrix}$$

Si deve fare attenzione ad introdurre i valori esatti dei parametri (N, M, P) e degli elementi delle due matrici A e B.

In questo calcolo N vale 2 (numero delle righe della matrice A), M vale 2 (numero delle colonne della matrice B) e P vale 3 (numero delle colonne di A e numero delle righe di B); inoltre, quando si introducono da tastiera le matrici A e B, occorre battere il valore di ogni elemento procedendo per riga: ad esempio, per la matrice A si batteranno i numeri 1, 2, 1, 0, 1, 1.

Tenendo presenti le indicazioni ora fornite, può passare ad eseguire il programma.

Batta quindi il comando BASIC **RUN** ed introduca con ordine nel computer i dati previsti.

Sullo schermo comparirà:

```
INTR. N, M, P ? 2, 2, 3
INTR. ELEM. A ? 1
INTR. ELEM. A ? 2
INTR. ELEM. A ? 1
INTR. ELEM. A ? 0
INTR. ELEM. A ? 1
INTR. ELEM. A ? 1
INTR. ELEM. B ?-1
INTR. ELEM. B ? 0
INTR. ELEM. B ? 1
INTR. ELEM. B ? 1
INTR. ELEM. B ? 3
INTR. ELEM. B ? 2
- - - - - - - - - - - - - - - - - - - - -
C(1,1)=4
```

```
----------------------
C (1,2)=4
----------------------
C (2,1)=4
----------------------
C (2,2)=3

READY
```

Come si può rilevare, i quattro elementi costituenti la matrice risultante C (di due righe e due colonne) sono visualizzati uno per volta sulle linee dello schermo.

In particolare, si ha:

C (1,1)=4: valore della prima riga e della prima colonna;
C (1,2)=4: valore della prima riga e della seconda colonna;
C (2,1)=4: valore della seconda riga e della prima colonna;
C (2,2)=3: valore della seconda riga e della seconda colonna;

per cui la matrice ottenuta, come già precedentemente precisato, risulta:

$$C(2,2)=\begin{pmatrix} 4 & 4 \\ 4 & 3 \end{pmatrix}$$

Il programma del prodotto di due matrici, prima codificato, è un utile strumento per effettuare una serie di altri calcoli utili e precisamente i seguenti:

- somma algebrica di P numeri;
- somma algebrica dei quadrati di P numeri;
- media aritmetica o ponderata di P numeri.

E' sufficiente a questo scopo introdurre, sempre utilizzando il programma codificato nella *fig. 3*, come matrice A un vettore di P elementi (costituito da P colonne e da una sola riga, cosiddetto "vettore-riga") e come matrice B un vettore di N elementi (ma costituito da P righe e da una sola colonna, cosiddetto "vettore-colonna"); la matrice C sarà formata da un solo elemento, che è appunto il risultato del prodotto di un vettore-riga A per un vettore-colonna B.

Quando il programma della *fig. 3* richiede di introdurre i parametri N, M, P, Lei introdurrà al posto di N e di M il semplice valore 1 ed al posto di P il valore numerico che rappresenta il numero dei dati su cui si vuole operare, secondo i tre tipi di calcoli prima citati e che ora vedremo.

SOMMA ALGEBRICA DI P NUMERI

Supponga, per esempio, di dover eseguire la semplice somma algebrica di seguito riportata:

33-44+5-60+77-800+1000+2000

il cui risultato è 2211.

Dovrà introdurre i parametri:

N=1 (tante sono le righe del vettore A)
M=1 (tante sono le colonne del vettore B)
P=8 (tanti sono gli elementi dei vettori A oppure B, ossia i numeri da sommare).

Per il vettore-riga A i dati da sommare devono essere introdotti **tutti con il segno +** (ricordi che il segno + può anche essere omesso), come di seguito indicato:

A(1,1)= 33
A(1,2)= 44
A(1,3)= 5
A(1,4)= 60
A(1,5)= 77
A(1,6)= 800
A(1,7)= 1000
A(1,8)= 2000

Per il vettore-colonna B dovranno invece essere introdotti i seguenti dati unitari (ognuno con l'appropriato segno + oppure - , a seconda se il

corrispondente numero da sommare è positivo o negativo):

```
B (1,1)= 1
B (2,1)=-1
B (3,1)= 1
B (4,1)=-1
B (5,1)= 1
B (6,1)=-1
B (7,1)= 1
B (8,1)= 1
```

Come si può rilevare, i dati unitari di B (2,1), B (4,1) e B (6,1) sono preceduti dal segno - , in quanto si riferiscono al secondo, al quarto ed al sesto numero da sommare i quali sono negativi (-44, -60, -800).

La sequenza completa di esecuzione del programma sarà quindi la seguente:

```
RUN

INTR. N, M, P ?  1, 1, 8
INTR. ELEM. A ?  33
INTR. ELEM. A ?  44
INTR. ELEM. A ?  5
INTR. ELEM. A ?  60
INTR. ELEM. A ?  77
INTR. ELEM. A ?  800
INTR. ELEM. A ?  1000
INTR. ELEM. A ?  2000
INTR. ELEM. B ?  1
INTR. ELEM. B ? -1
INTR. ELEM. B ?  1
INTR. ELEM. B ? -1
INTR. ELEM. B ?  1
INTR. ELEM. B ? -1
INTR. ELEM. B ?  1
INTR. ELEM. B ?  1
----------------------
C (1,1)=2211
```

Il risultato ottenuto (2211), come si noterà, risulta eguale a quello precedentemente indicato.

In effetti, il calcolo matematico del prodotto del vettore-riga:

$$A(1,8)=(33,\ 44,\ 5,\ 60,\ 77,\ 800,\ 1000,\ 2000)$$

per il vettore-colonna:

$$B(8,1)=\begin{pmatrix}1\\-1\\1\\-1\\1\\-1\\1\\1\end{pmatrix}$$

eseguito moltiplicando e sommando gli elementi corrispondenti di entrambi i vettori, porta allo stesso risultato ottenuto con il programma della *fig. 3*.

Infatti:

$$C(1,1)=A(1,8)*B(8,1)=(33,\ 44,\ 5,\ 60,\ 77,\ 800,\ 1000,\ 2000)*\begin{pmatrix}1\\-1\\1\\-1\\1\\-1\\1\\1\end{pmatrix}=$$

$$=33*1+44*(-1)+5*1+60*(-1)+77*1+800*(-1)+1000*1+2000*1=2211$$

SOMMA ALGEBRICA DEI QUADRATI DI P NUMERI

Per eseguire questo calcolo, si devono immettere nel computer sempre gli stessi parametri di partenza e successivamente introdurre per il vettore-riga A i dati numerici di cui si vuole calcolare la somma dei quadrati, e per il vettore B gli stessi dati di A, ma con gli appropriati segni degli elementi.

Per esempio, per calcolare la seguente somma algebrica:

$$3^2-2^2+8^2=69$$

Lei dovrà introdurre dapprima i parametri:

$$N=1$$
$$M=1$$
$$P=3$$

e successivamente i seguenti vettori-riga e colonna:

```
A(1,1)= 3
A(1,2)= 2
A(1,3)= 8
B(1,1)= 3
B(2,1)=-2
B(3,1)= 8
```

per avere il risultato:

$$C(1,1)=A(1,3)*B(3,1)=(3,2,8)*\begin{pmatrix}3\\-2\\8\end{pmatrix}=3^2-2^2+8^2=69$$

L'esecuzione del programma sarà:

```
RUN

INTR. N, M, P ?  1, 1, 3
INTR. ELEM. A ?  3
INTR. ELEM. A ?  2
INTR. ELEM. A ?  8
INTR. ELEM. B ?  3
INTR. ELEM. B ? -2
INTR. ELEM. B ?  8
- - - - - - - - - - - - - - - - - - - - - -
C(1,1)=69
```

MEDIA ARITMETICA O PONDERATA DI P NUMERI

Per calcolare, infine, la media aritmetica o ponderata di P numeri, occorre che Lei, al posto del vettore-colonna B, introduca appropriati "pesi".

Se, per esempio, vuole calcolare la media aritmetica semplice dei quattro numeri: 5Ø, 1ØØ, 15Ø, 2ØØ, dovrà introdurre i seguenti parametri:

```
N=1
M=1
P=4
```

dati: A=(50, 100, 150, 200)

$$\text{pesi: } B=\begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$$

per ottenere sullo schermo il risultato C(1,1)=125.

Occorre tenere presente che il "peso" di ognuno degli elementi A, per calcolare la media aritmetica semplice, è 1/4 se 4 è il numero dei dati oppure 1/n se il numero dei dati è **n**.

Infatti, la media aritmetica semplice M dei quattro numeri 5Ø, 1ØØ, 15Ø, 2ØØ si può scrivere nel seguente modo:

$$M=\frac{50+100+150+200}{4}=\frac{1}{4}*50+\frac{1}{4}*100+\frac{1}{4}*150+\frac{1}{4}*200=125$$

In questa espressione aritmetica il fattore 1/4 costituisce il "peso" di ogni dato da mediare con gli altri.

Nell'esecuzione pratica del programma si introdurrà come "peso" il valore numerico 0,25 (1/4=0,25).

Il programma sarà attuato nel modo seguente:

```
RUN

INTR. N, M, P ?  1, 1, 4
INTR. ELEM. A ?  50
INTR. ELEM. A ?  100
INTR. ELEM. A ?  150
INTR. ELEM. A ?  200
INTR. ELEM. B ?  0.25
INTR. ELEM. B ?  0.25
INTR. ELEM. B ?  0.25
INTR. ELEM. B ?  0.25
----------------------
C(1,1)=125
```

Esercizio 2

Come secondo esercizio di programmazione sull'uso dell'istruzione **FOR... NEXT...** proveremo a programmare il problema descritto nella lezione *Analisi e Programmazione 3*, il quale riguardava l'ordinamento crescente di un numero N di valori numerici.

Nella *fig. 4* è riportato il flow-chart di tale programma e nella *fig. 5* la relativa codifica in linguaggio BASIC.

Il programma in questione prevede l'immissione da tastiera del parametro N, che rappresenta il numero dei valori numerici da ordinare in modo crescente.

Naturalmente, i valori da immettere possono essere casualmente piú piccoli o piú grandi dei primi battuti, perché sarà appunto compito del programma ordinarli in modo crescente.

La parte del programma che va dalla linea 2Ø alla linea 35 realizza l'immissione dei dati da tastiera.

Questi dati sono memorizzati nel vettore di nome A dimensionato N, dove N è il numero degli elementi numerici da ordinare.

Nelle istruzioni 25 e 35 viene definito un ciclo di elaborazione che verrà ripetuto N volte.

Ad ogni giro del ciclo saranno eseguite le istruzioni 27 e 3Ø, le quali faranno comparire sullo schermo la richiesta d'ingresso dati, come riportato qui sotto:

```
A(1)
?
A(2)
?
A(3)
?
.
.
.
```

Il primo elemento numerico da ordinare sarà memorizzato come A(1), il secondo elemento come A(2) e cosí via di seguito.

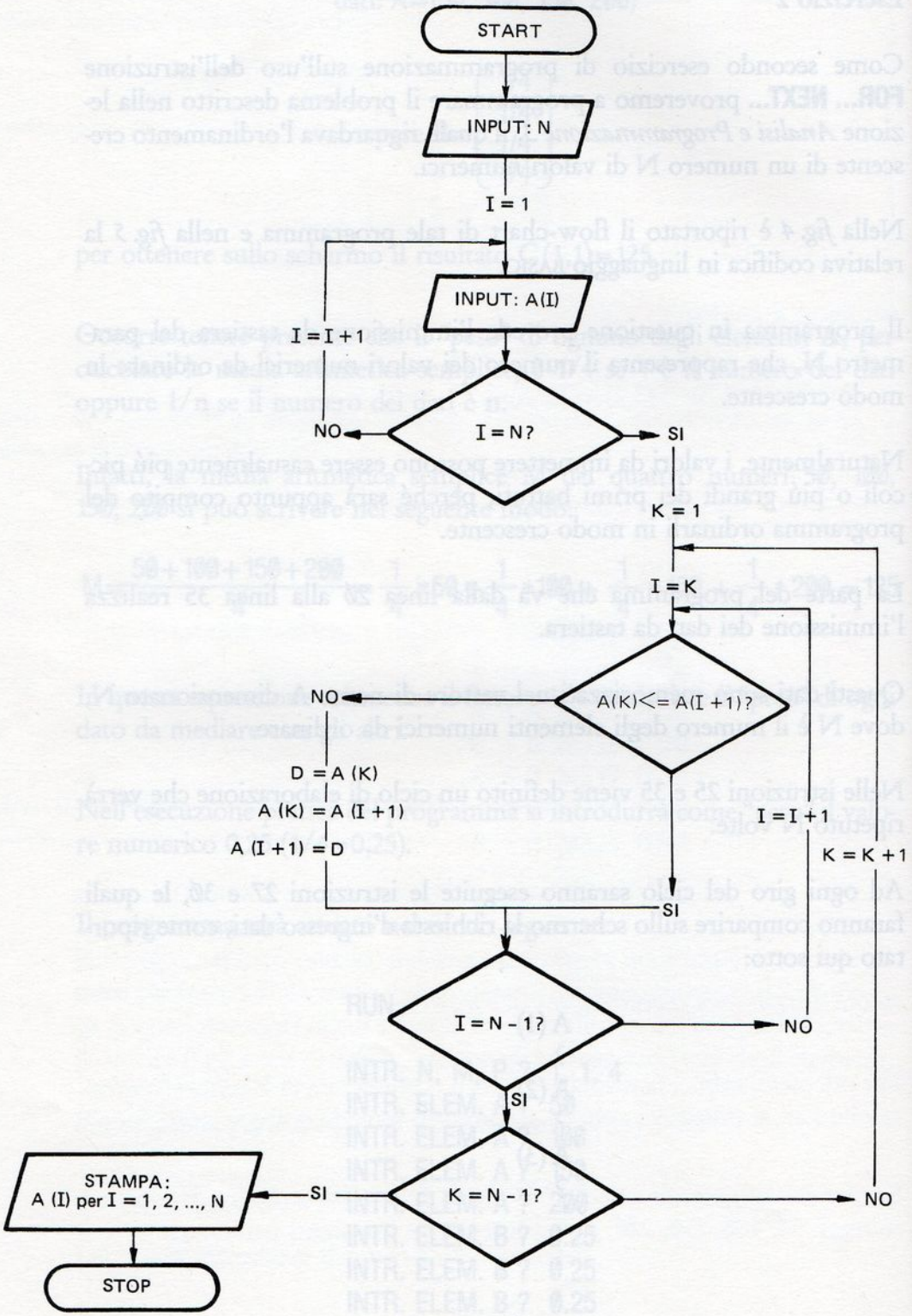


Fig. 4 – Flow-chart del programma che ordina una serie di N numeri.

```
10 REM******************
15 REM* ORDINAMENTO  *
17 REM* CRESCENTE DI *
18 REM*  N NUMERI    *
19 REM******************
20 INPUT"NUM.DATI:";N
23 DIM A(N)
25 FOR I=1 TO N
27 PRINT"A(";I;")
30 INPUT A(I):PRINT
35 NEXT I
40 FOR K=1 TO N-1
45 FOR I=K TO N-1
50 IF A(K)<=A(I+1) THEN 70
55 D=A(K)
60 A(K)=A(I+1)
65 A(I+1)=D
70 NEXT I
75 NEXT K
80 PRINT"SUCCESSIONE ORDINATA:":PRINT
85 FOR I=1 TO N
90 PRINT A(I)
95 NEXT I
100 END
```

Fig. 5 - Codifica del programma di ordinamento crescente di N numeri.

La parte del programma che va dalla linea 40 alla linea 75 realizza il vero e proprio ordinamento dei valori numerici.

In questa parte del programma sono presenti due cicli annidati: quello esterno è definito dalle istruzioni 40 e 75, quello interno dalle istruzioni 45 e 70.

Il procedimento di ordinamento crescente adottato contempla un numero di confronti pari a:

$$\frac{N*(N-1)}{2}$$

Per esempio, nel caso in cui i numeri da ordinare siano i seguenti sei elementi numerici:

3.14, 1, 2, 50, 63.3, 6

il numero dei confronti da fare sarebbe:

$$\frac{6*5}{2}=15$$

Il procedimento adottato non è l'unico possibile; esistono infatti procedimenti (o algoritmi logici) piú veloci, che prevedono un numero di confronti anche inferiore.

Conclusa la descrizione del programma, può ora immetterlo nel computer.

Per verificare l'efficienza del programma in questione, supponga di ordinare in modo crescente i sei numeri citati prima, ossia:

3.14, 1, 2, 50, 63.3, 6

La sequenza di esecuzione del programma sarà:

```
RUN
NUM. DATI ? 6
A (1)
?  3.14
A (2)
?  1
A (3)
?  2
A (4)
?  50
A (5)
?  63.3
A (6)
?  6
SUCCESSIONE ORDINATA
 1
 2
 3.14
 6
 50
 63.3
```

Ottenuto esito positivo dalla verifica del programma, può registrarlo sul lato B della seconda cassetta da Lei ricevuta, o su un'altra qualsiasi cas-

setta, assegnando al programma stesso un apposito nome.

Per la corretta registrazione, segua la procedura piú volte indicata nelle precedenti lezioni.

2. – Le istruzioni READ e DATA

Queste due istruzioni **vanno sempre usate insieme.**

L'istruzione **READ** istruisce il computer a prelevare dati numerici o stringa ed a depositarli nelle zone di memoria contraddistinte appropriatamente con nomi di variabili.

I valori da attribuire alle variabili della **READ** sono forniti dall'istruzione **DATA**, che funge cosí da archivio interno di dati.

Vedremo ora una pratica applicazione di queste due istruzioni, realizzando un programma di utilità di tipo fiscale: il calcolo dell'imposta sul reddito delle persone fisiche, altrimenti denominata IRPEF.

PROGRAMMA CALCOLO IMPOSTA LORDA IRPEF

Come Lei forse saprà, il calcolo dell'IRPEF si fa una volta all'anno, al momento della dichiarazione dei redditi, sull'apposito modulo 740 messo a disposizione dei contribuenti da parte del governo italiano.

Non sempre il calcolo dell'IRPEF risulta agevole, perciò sarà cosa certamente utile elaborare un programma che sia d'aiuto nella compilazione di almeno una parte della dichiarazione stessa: esattamente il quadro N del riepilogo IRPEF.

Si tratta di calcolare l'imposta lorda sul reddito imponibile in base ai cosiddetti "scaglioni di reddito" (il reddito imponibile è la cifra riportata nel corrispondente rigo del quadro N del modulo 740, cifra che è a sua volta il risultato della detrazione del totale oneri deducibili dal totale reddito complessivo).

Calcolato il reddito imponibile, si individua lo scaglione di reddito entro cui esso è compreso.

Quindi si calcola l'imposta lorda con la formula seguente:

$$IL = T(I) + [R - S(I)]\ A(I)$$

dove:

IL = imposta lorda da segnare nel relativo rigo del quadro N;
T(I) = imposta dovuta sull'ammontare massimo dello scaglione di reddito;
S(I) = valore minore dell'ultimo scaglione di reddito;
R = valore del reddito imponibile compreso tra S(I) e S(I+1);
A(I) = aliquota relativa all'ultimo scaglione di reddito.

Gli scaglioni di reddito riportati sul modulo 740 e relativi all'anno fiscale 1983 sono quelli raffigurati nella *fig. 6*.

Numero scagl. (I)	Scaglioni di reddito	Aliq. %	Imposta dovuta sull'ammontare massimo dello scaglione	Imposta dovuta sui redditi intermedi compresi negli scaglioni
1	0 ÷ 11.000.000	18	1.980.000	18% sull'intero importo
2	11.000.001 ÷ 24.000.000	27	5.490.000	1.980.000 + 27% su eccedente 11.000.000
3	24.000.001 ÷ 30.000.000	35	7.590.000	5.490.000 + 35% " 24.000.000
4	30.000.001 ÷ 38.000.000	37	10.550.000	7.590.000 + 37% " 30.000.000
5	38.000.001 ÷ 60.000.000	41	19.570.000	10.550.000 + 41% " 38.000.000
6	60.000.001 ÷ 120.000.000	47	47.770.000	19.570.000 + 47% " 60.000.000
7	120.000.001 ÷ 250.000.000	56	120.570.000	47.770.000 + 56% " 120.000.000
8	250.000.001 ÷ 500.000.000	62	275.570.000	120.570.000 + 62% " 250.000.000
9	500.000.001 ÷ OLTRE	65		275.570.000 + 65% " 500.000.000

Fig. 6 – Tabella degli scaglioni di reddito relativi all'anno fiscale 1983.

Il programma che si accingiamo a formulare è relativo quindi alle norme fiscali del 1984; eventuali modifiche governative alla curva dei redditi comportano parziali correzioni del programma.

Come si vede dalla *fig. 6*, relativamente al primo scaglione di reddito (I=1) si ha S(1)=0 (reddito nullo) fino a 11.000.000, per il secondo scaglione (I=2) si ha S(2)=11.000.001 fino a 24.000.000, per il terzo scaglione (I=3) si ha S(3)=24.000.001 fino a 30.000.000 e cosí via di seguito, sino al nono scaglione (I=9) in cui S(9)=500.000.001 ed oltre.

Per il buon funzionamento della formula:

$$IL = T(I) + [R - S(I)]\ A(I)$$

occorre inoltre supporre che per I=1 essa si riduca all'espressione:

$$IL = R \cdot A(1) = R \cdot 18\%$$

Verifichiamo questa formula eseguendo a titolo di esempio il calcolo dell'imposta lorda dovuta sul reddito imponibile di 16.000.000.

In questo caso il reddito (16 milioni) è compreso nel secondo scaglione (I=2), quindi la formula diventa;

$$\begin{aligned} IL &= T(2) + (R - S(2) \cdot A(2) = \\ &= 1.980.000 + (16.000.000 - 11.000.000) \times 27\% = \\ &= 1.980.000 + (5.000.000) \times 0{,}27 = 3.330.000 \end{aligned}$$

Analogamente, per un reddito imponibile di 50.000.000 si ha:

$$\begin{aligned} IL &= 10.550.000 + (50.000.000 - S(5)) \cdot A(5) = \\ &= 10.550.000 + (50.000.000 - 38.000.000) \times 41\% = \\ &= 10.550.000 + (12.000.000) \times 0{,}41 = 15.470.000 \end{aligned}$$

Sulla base di quanto sinora esposto, nella *fig. 7* è stato tracciato il flow-chart del programma che calcola l'imposta lorda IRPEF per l'anno 1984, mentre nella *fig. 8* è riportata la relativa codifica in linguaggio BASIC.

Tale programma, a cui Lei può assegnare un nome a piacere (per esempio, "IRPEF 84"), può essere provato con redditi diversi e registrato sul nastro magnetico a Sua disposizione.

Come di consueto, prima di battere sulla tastiera del computer il programma in questione, è opportuno esaminarne le fasi operative piú importanti.

Le istruzioni 2Ø e 3Ø definiscono un ciclo di programma che viene eseguito nove volte, ossia tante volte quanti sono gli scaglioni (I) di reddito relativo all'anno fiscale 1983/1984.

Ad ogni giro del ciclo viene eseguita l'istruzione 25, mediante la quale vengono memorizzati, nelle locazioni di memoria chiamate S(I), A(I), T(I), i dati riportati nelle istruzioni DATA (numerate da 15Ø a 2ØØ.

In altre parole, a tali locazioni verranno attribuiti i valori numerici in-

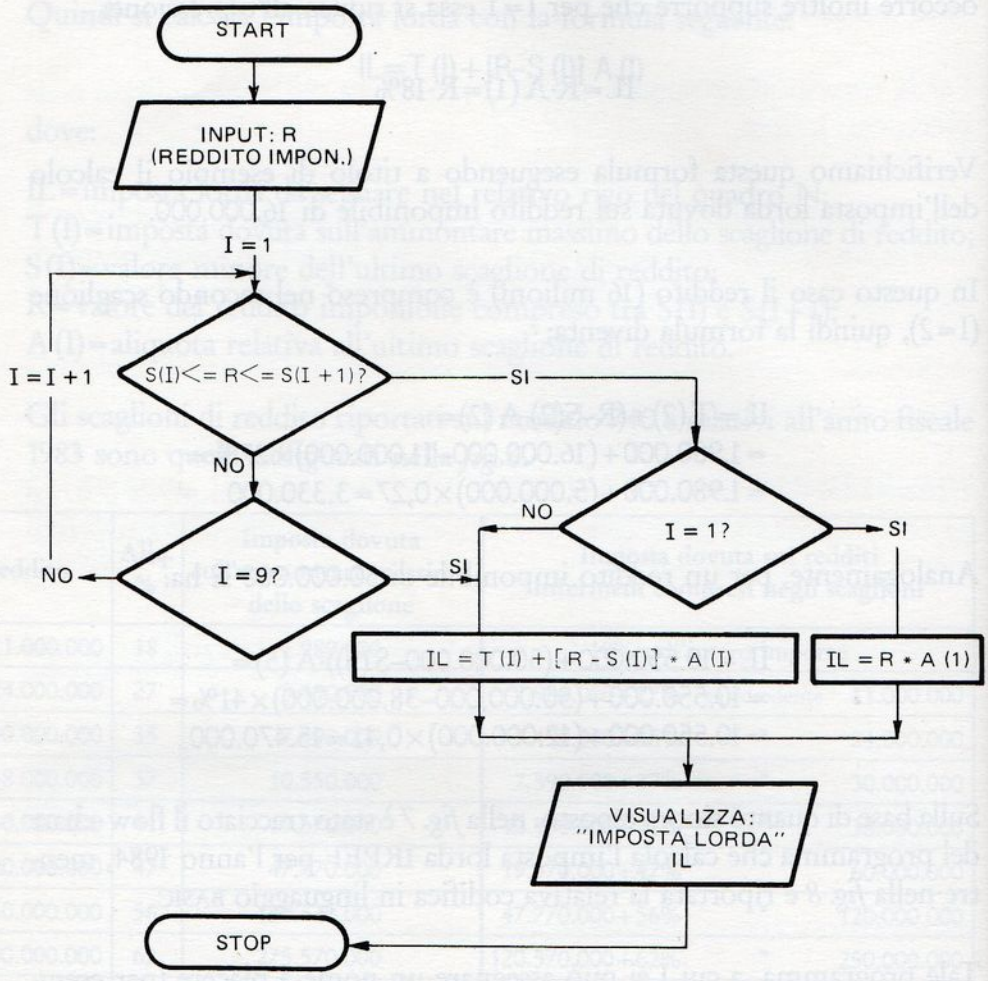


Fig. 7 - Flow-chart del programma "IRPEF 84".

dicati di seguito.

Per I=1
S(I)=Ø, A(I)=Ø.18, T(I)=Ø.

Per I=2
S(I)=11.ØØØ.ØØØ, A(I)=0.27, T(I)=1.98Ø.ØØØ,

e cosí via di seguito.

L'istruzione 35 provvede a pulire lo schermo e a posizionare il cursore in alto a sinistra.

Ricordi che per ottenere ciò occorre premere il tasto **SHIFT** e, mantenendolo premuto, abbassare il tasto **CLR | HOME**, il quale viene appunto raffigurato sullo schermo con un piccolo cuore.

```
10 REM****************
12 REM*   CALCOLO     *
15 REM*IMPOSTA LORDA *
17 REM*    IRPEF 84   *
18 REM****************
20 FOR I=1 TO 9
25 READ S(I),A(I),T(I)
30 NEXT I
35 PRINT"□"
40 PRINT"CALCOLO IMPOSTA LORDA DEL REDDITO:"
45 INPUT R
50 PRINT:PRINT"----------------------------------------"
55 I=1
60 IF R>=S(I) AND R<=S(I+1) THEN 80
65 IF I=9 THEN 85
70 I=I+1
75 GOTO 60
80 IF I=1 THEN 95
85 IL=T(I)+((R-S(I))*A(I)
90 GOTO 100
95 IL=R*A(1)
100 PRINT:PRINT"■";IL"■"
110 STOP
120 GOTO 35
125 END
150 DATA 0,0.18,0
160 DATA 11000000,0.27,1980000
170 DATA 24000000,0.35,5490000
175 DATA 30000000,0.37,7590000
180 DATA 38000000,0.41,10550000
185 DATA 60000000,0.47,19570000
190 DATA 120000000,0.56,47770000
195 DATA 250000000,0.62,120570000
200 DATA 500000000,0.65,275570000
```

Fig. 8 – Codifica in linguaggio BASIC *del programma "IRPEF 84".*

Il valore del reddito imponibile R viene introdotto nel calcolatore mediante l'istruzione 45.

L'istruzione 5Ø provvede a visualizzare sullo schermo quaranta trattini orizzontali ottenuti con il tasto — (ossia pari alla larghezza di stamper del computer C64), il cui scopo è di separare il messaggio

"CALCOLO IMPOSTA LORDA DEL REDDITO"

dal risultato del calcolo stesso.

Se il programma in questione viene eseguito con il computer VIC-20 anziché con il C64, le istruzioni 4Ø e 5Ø del listato della *fig. 8* devono essere opportunamente modificate per tenere conto della diversa larghezza di stampa (ventidue caratteri per il VIC-20).

Pertanto, nel caso in cui si operi con il VIC-20, le istruzioni citate dovranno essere cosí modificate:

```
⋮
40 PRINT "CALCOLO IMPOSTA LORDA"
43 PRINT "DEL REDDITO"
⋮
50 PRINT:PRINT "_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _"
⋮
```

L'istruzione di diramazione 6Ø si chiede se la variabile R (valore del reddito) è contemporaneamente maggiore o uguale a S(I) e minore o uguale a S(I).

In caso affermativo l'elaborazione prosegue per l'istruzione 8Ø, dove un'altra istruzione di diramazione chiede se I=1; se cosí è, viene eseguita l'istruzione 95, che provvede a calcolare l'imposta lorda con la formula ridotta: IL=R*A(1).

Per gli altri valori di I l'imposta lorda viene calcolata con la formula riportata nell'istruzione 85.

L'istruzione 1ØØ, invece, fa visualizzare sullo schermo il valore dell'imposta lorda (di nome IL) in formato negativo (cifre luminose chiare su fondo scuro sul VIC-20; cifre scure su fondo chiaro sul C64). A questo scopo viene appunto utilizzato il tasto **RVS | ON**, mentre il tasto **RVS | OFF** permette di ripristinare le condizioni di visualizzazione standard.

Il tasto operativo **RVS | ON**, quando è inserito fra due caratteri " (virgolette), appare raffigurato sullo schermo come una lettera **R**, mentre il tasto **RVS | OFF** viene rappresentato con il simbolo ■.

Ricordi che i tasti in questione sono attivati dal tasto **CTRL**, il quale deve essere mantenuto premuto mentre si battono i tasti **RVS | ON** e **RVS | OFF**.

Come già precisato, gli scaglioni di reddito sono stati memorizzati in fondo al programma nelle istruzioni di archivio **DATA**, che vanno dal numero 15Ø al numero 2ØØ.

E' chiaro che questi scaglioni, se dovessero essere modificati, andrebbero corretti nel programma insieme alle corrispondenti aliquote nuove.

Questo programma potrebbe essere migliorato (con l'aggiunta di altre istruzioni) in modo da consentire di effettuare le cosiddette "detrazioni d'imposta" che, come Lei saprà, sono sempre previste nella dichiarazione dei redditi.

A tal fine basterebbe aggiungere nel programma precedente, sia prima sia dopo l'individuazione dello scaglione d'imposta entro cui è compreso il generico reddito R, apposite istruzioni che effettuino su quest'ultimo delle sottrazioni di qualsiasi importo (introdotto eventualmente da tastiera con l'istruzione **INPUT**).

Terminato l'esame del programma, può ora immetterlo nel computer battendo con cura le istruzioni che lo costituiscono.

Per collaudare il programma in questione, possiamo provare a calcolare l'imposta lorda di un reddito di L. 16.000.000.

L'impostazione del programma sarà la seguente:

```
RUN

CALCOLO IMPOSTA LORDA DEL REDDITO:
? 16.000.000
-----------------------------------------
3.330.000
BREAK IN 110
READY
```

Ossia l'imposta lorda dovuta per il reddito di 16.000.000 di lire risulta pari a L. 3.300.000, come visto nell'esempio prima considerato.

Passiamo ora ad esaminare il modo in cui si programma il computer per la riproduzione di immagini colorate.

Poiché le istruzioni per ottenere l'inserimento dei colori sullo schermo sono sensibilmente diverse a seconda che si operi con il C64 o con il VIC-20, esse verranno trattate separatamente.

A seconda quindi del computer in Suo possesso (C64 oppure VIC-20), dovrà fare riferimento ovviamente al relativo capitolo.

3. – Programmazione a colori sul C64

La raffigurazione del colore sullo schermo del C64 può essere effettuata in due modi.

Il primo, che si può definire "manuale", consiste nel compiere operazioni particolari con la tastiera, le quali consentono la visualizzazione di sedici diversi colori per i caratteri.

Otto di tali colori sono ottenuti, come precisato in precedenza, premendo il tasto CTRL ed uno dei tasti colore, cioè uno dei tasti numerati da 1 a 8, sulla cui superficie anteriore sono riportate le seguenti abbreviazioni in inglese:

1 – BLK = nero	5 – PUR = porpora
2 – WHT = bianco	6 – GRN = verde
3 – RED = rosso	7 – BLU = blu
4 – CYN = blu-verde	8 – YEL = giallo

Premendo il tasto Commodore C= unitamente ad uno dei tasti numerati da 1 a 8, si ottengono i seguenti altri otto colori per il tasto:

1 = arancio	5 = grigio 2
2 = marrone	6 = verde chiaro
3 = rosso chiaro	7 = azzurro
4 = grigio 1	8 = grigio 3

Quando da tastiera si vuole fare apparire sullo schermo uno o piú caratteri con un certo colore, quest'ultimo viene impostato prima pigiando il tasto CTRL oppure il tasto C= e quindi il tasto del colore prescelto.

Per esempio, se si vuole raffigurare sullo schermo la lettera A in rosso su sfondo blu (colore standard del C64), si esegue la seguente istruzione BASIC immediata:

PRINT " CTRL → RED **A"** RETURN

Con i tasti CTRL → RED viene impostato sullo schermo il colore

rosso dei caratteri (la freccia → sta ad indicare, come di consueto, che il primo tasto di controllo CTRL viene tenuto abbassato mentre si preme il secondo tasto RED).

Con l'istruzione **PRINT**, ora eseguita, il computer viene predisposto in modo che i successivi caratteri impostati dopo CTRL → RED siano visualizzati in rosso.

Come ha potuto rilevare battendo il tasto RED, mantenendo premuto il tasto CTRL, sullo schermo è apparso il simbolo £ (lira sterlina) in rappresentazione inversa.

Infatti l'impostazione dei colori nell'interno di una stringa, ossia all'interno dei caratteri " (virgolette) di una **PRINT**, assume un carattere "stenografico", nel senso che, mentre da tastiera si battono i tasti di impostazione del colore (CTRL → [COLORE]) oppure (C= → [COLORE]), sullo schermo appaiono "altri" simboli che praticamente costituiscono la rappresentazione stenografica dei colori stessi.

Per meglio comprendere ciò, nella *fig. 9* è rappresentata una tabella riportante le corrispondenze tra i vari simboli che appaiono sullo schermo e che vengono riprodotti dalla stampante, quando si imposta un comando del colore all'interno di una stringa.

Come ha già avuto modo di constatare nelle precedenti lezioni, vi sono anche altri comandi, oltre quelli del colore, che, impostati all'interno di una stringa, fanno comparire sullo schermo e riprodurre dalla stampante simboli stenografici, come si vede nella *fig. 10*.

Dopo avere impostato sullo schermo il colore dei caratteri, ad esempio il rosso come visto nell'istruzione immediata prima eseguita, occorre reimpostare i colori se si intende raffigurare i caratteri successivi con altri colori.

Per esempio, se si desidera raffigurare gli otto numeri 1, 2, 3, 4, 5, 6, 7, 8 nella prima serie di otto colori disponibili su tastiera, si può eseguire la seguente istruzione BASIC:

PRINT " CTRL → BLK 1 CTRL → WHT 2 CTRL → RED 3
CTRL → CYN 4 CTRL → PUR 5 CTRL → GRN 6
CTRL → BLU 7 CTRL → YEL 8"

Effettuando con il tasto RETURN questa istruzione **PRINT**, sullo scher-

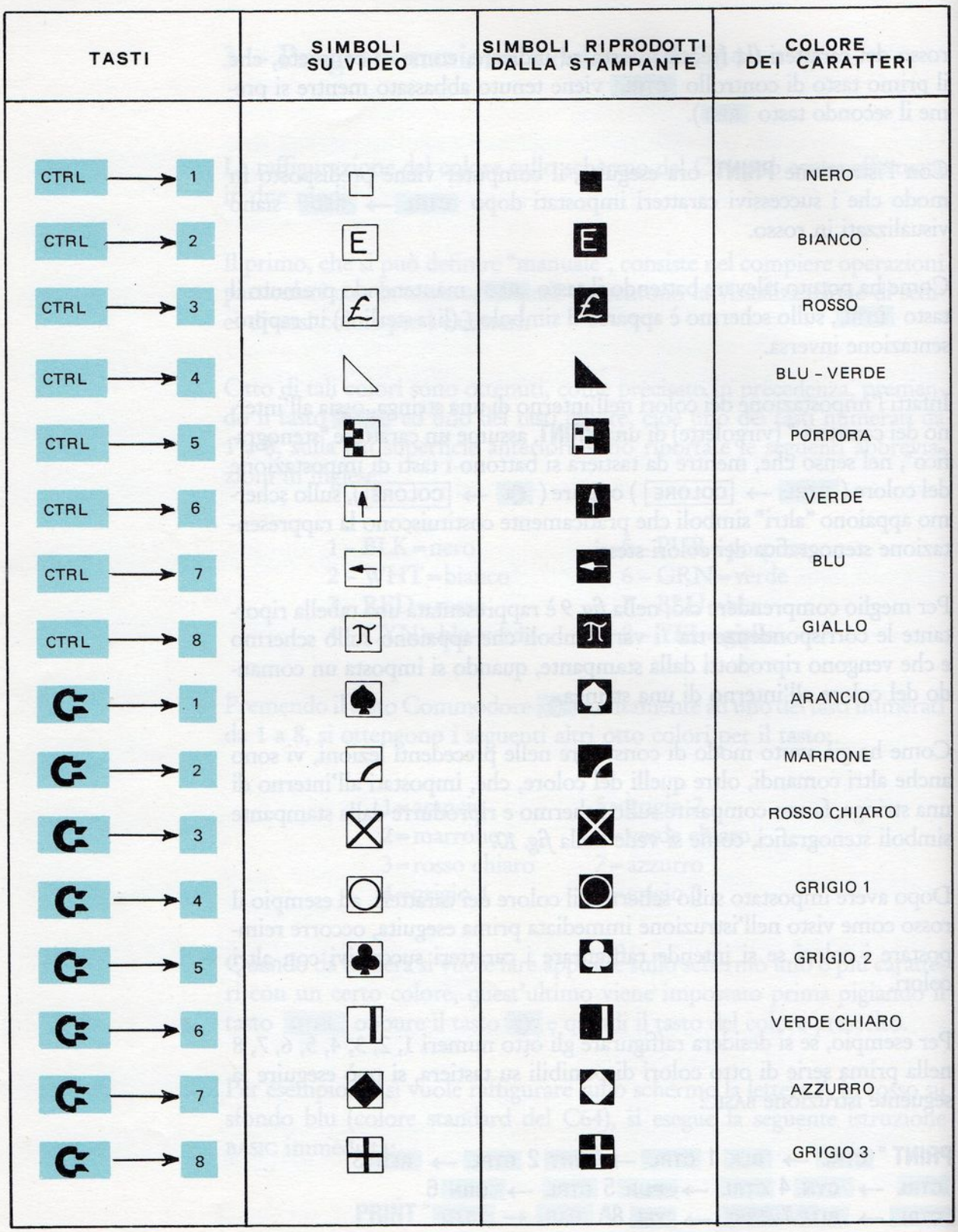


Fig. 9 – Rappresentazione grafica ottenuta sullo schermo e riprodotta dalla stampante quando vengono impostati i colori all'interno di una stringa nel C64.

SIMBOLI SU VIDEO	SIMBOLI RIPRODOTTI DALLA STAMPANTE	COMANDI	FUNZIONE
♥	♥	SHIFT → CLR || HOME	CANCELLAZIONE DELLO SCHERMO E POSIZIONAMENTO DEL CURSORE IN ALTO A SINISTRA
●	○	SHIFT → ⇑ CRSR ⇓	CURSORE IN ALTO
Q	Q	⇑ CRSR ⇓	CURSORE IN BASSO
┘	┘	⇐ CRSR ⇒	CURSORE A DESTRA
▌	▌	SHIFT → ⇐ CRSR ⇒	CURSORE A SINISTRA
R	R	CTRL → RVS || ON	CARATTERE IN CAMPO INVERSO (NEGATIVO)
▬	▬	CTRL → RVS || OFF	RITORNO DAL CAMPO INVERSO AL MODO NORMALE
S	S	CLR || HOME	POSIZIONAMENTO DEL CURSORE IN ALTO A SINISTRA

Fig. 10 – Rappresentazione grafica di comandi impostati all'interno di una stringa nel computer C64.

mo apparirà la seguente stringa colorata di numeri:

1 2 3 4 5 6 8
(NERO) (BIANCO) (ROSSO) (VERDE-BLU) (PORPORA) (VERDE) (GIALLO)

Soltanto il numero 7 non si vede. Perché? Il motivo è semplice: il 7 ha colore blu su fondo blu e quindi non appare. Bisogna infatti fare attenzione che con questo criterio di colorazione lo schermo rimane sempre blu, che è appunto il suo colore standard, il quale può essere variato in concomitanza con i colori dei caratteri, come vedremo successivamente.

Dopo l'esecuzione dell'istruzione **PRINT** vista prima, il colore dei caratteri riprodotti sullo schermo rimane impostato sul giallo, ossia sul colore corrispondente all'ultimo carattere della stringa (8).

Questa impostazione dei caratteri (giallo su fondo blu) perdura finché non viene nuovamente ripristinata l'impostazione standard che prevede

il carattere azzurro su fondo blu con margine azzurro.

Per ripristinare l'impostazione standard è sufficiente premere il tasto **RUN | STOP** e, mantenendolo premuto, battere il tasto **RESTORE**. Con questa operazione lo schermo viene ripulito dalle scritte, ma il programma eventualmente presente nella memoria principale non viene cancellato.

A questo punto, volendo visualizzare gli otto numeri 1, 2, 3, 4, 5, 6, 7, 8 nella seconda serie di otto colori disponibili su tastiera, si può eseguire la seguente istruzione BASIC:

```
PRINT" C= → 1 1 C= → 2 2 C= → 3 3
       C= → 4 4 C= → 5 5 C= → 6 6
       C= → 7 7 C= → 8 8"
```

Premendo il tasto **RETURN**, sullo schermo apparirà la seguente, seconda stringa colorata:

1 (ARANCIO) 2 (MARRONE) 3 (ROSSO CHIARO) 4 (GRIGIO 1) 5 (GRIGIO 2) 6 (VERDE CHIARO) 7 (AZZURRO) 8 (GRIGIO 3)

Dopo l'esecuzione di questa istruzione il colore dei caratteri rimane impostato sul grigio 3, che è appunto il colore dell'ultimo carattere battuto (8). Per ritornare alla condizione standard, occorre premere il tasto **C=** e, mantenendolo premuto, battere il tasto **7** (colore azzurro) oppure, come già specificato, premere contemporaneamente i tasti **RUN | STOP** → **RESTORE**.

Ora che si è visto come si fa a cambiare i colori dei caratteri sullo schermo costantemente blu, possiamo verificare come è possibile variare i colori dello schermo stesso (fondo) e del margine (bordo o cornice), servendosi di apposite istruzioni.

Nel computer C64 esistono due specifiche locazioni di memoria principale riservate all'impostazione dei colori del margine e dello schermo: precisamente, la **locazione 53280 per i colori del margine e la locazione 53281 per i colori dello schermo.**

Come già precisato in precedenza, per poter depositare i dati nella memoria principale del computer viene utilizzata l'istruzione **POKE.**

Questa istruzione è sempre seguita da due numeri: il primo è l'indirizzo di memoria (nel nostro caso 5328Ø oppure 53281); il secondo numero, separato dal primo da una virgola, è il codice relativo al colore prescelto.

Inserendo quindi, mediante l'istruzione **POKE**, un valore qualsiasi compreso fra Ø e 255 nelle locazioni di memoria prima citate, si possono ottenere diversi colori del margine e dello schermo.

In pratica, però, **vengono utilizzati soltanto i valori compresi fra Ø e 15**, in quanto gli altri valori numerici ripetono gli stessi colori.

Nella tabella che segue sono riportati i codici numerici da inserire (POKE) nelle locazioni di memoria 5328Ø e 53281 per ottenere ciascun colore del margine e dello schermo.

CODICE	COLORE	CODICE	COLORE
0	NERO	8	ARANCIO
1	BIANCO	9	MARRONE
2	ROSSO	10	ROSSO CHIARO
3	CELESTE	11	GRIGIO 1
4	PORPORA	12	GRIGIO 2
5	VERDE	13	VERDE CHIARO
6	BLU	14	AZZURRO
7	GIALLO	15	GRIGIO 3

Per esempio, se si desidera avere lo schermo di colore bianco, si deve eseguire la seguente istruzione immediata:

POKE 53281,1 RETURN

Il colore dello schermo sarà bianco, in quanto è stato inserito il valore 1, corrispondente appunto al colore bianco, nella locazione di memoria 53281 che controlla il colore dello schermo.

Se invece si desidera ottenere il margine di colore verde, occorre scrivere la seguente istruzione:

POKE 5328Ø,5 RETURN

Infatti il numero 5 corrisponde al colore verde.

Il semplice programma riportato nella *fig. 11* consente di visualizzare e

```
5 REM*COLORI SCHERMO E MARGINE*
10 PRINT"□"
15 FOR X=0 TO 15
20 PRINT TAB(10);"CODICE MARGINE=";X
25 POKE 53280,X :PRINT
30 FOR Y=0 TO  15
35 PRINT"CODICE SCHERMO=";Y
40 POKE 53281,Y
45 FOR T=1 TO 1000:NEXT T
50 NEXT Y
55 PRINT"□"
60 NEXT X
65 END
```

Fig. 11 – Programma per la visualizzazione dei colori dello schermo e del margine sul C64.

constatare personalmente tutte le diverse combinazioni di colori del margine e dello schermo.

In questo programma l'istruzione 15 assegna alla variabile X il valore iniziale Ø ed effettua un ciclo di programma che termina con l'istruzione 6Ø (**NEXT** X).

Ad ogni ciclo la variabile X viene aumentata di 1 ed il programma ritorna all'istruzione 25.

Il ciclo termima quando la variabile X (codice del margine) ha raggiunto il valore 15.

L'istruzione 25 immette nel byte di controllo del colore del margine, con indirizzo 53280, il valore assunto in quel momento dalla variabile X, che corrisponde ad un prefissato colore.

In modo analogo nell'istruzione 3Ø la variabile Y (colore dello schermo) viene incrementata di 1 ad ogni ciclo e l'istruzione 4Ø immette nel byte di controllo del colore dello schermo, con indirizzo 53281, il valore assunto in quel momento dalla variabile Y, che corrisponde ad uno dei quindici colori previsti.

L'istruzione **FOR... NEXT...**, riportata alla riga 45 del programma, costituisce una pausa temporale (ciclo di ritardo); in tal modo chi osserva lo schermo ha la possibilità di annotare qualche particolare combinazione di colori di suo interesse.

Qualora si intenda prolungare questa pausa, basta aumentare il valore

1000 della **FOR**, portandolo eventualmente a 2000.

In pratica, con l'istruzione 45 il computer si mette semplicemente a "contare" da 1 a 1000 o 2000 e, "contando", consente una piccola pausa a chi deve controllarlo.

Terminata la descrizione del listato della *fig. 11*, cancelli la memoria del computer mediante il comando BASIC **NEW**.

Dopo di ciò immetta il nuovo programma nel computer e lo esegua scrivendo il comando **RUN**.

Come potrà osservare, per ognuno dei quindici colori del margine compariranno i quindici diversi colori dello schermo.

La combinazione dei vari colori termina con lo schermo ed il margine di colore grigio.

Per ottenere la condizione di colore standard del video, prema, come già precisato, il tasto **RUN | STOP** e, mantenendolo premuto, abbassi il tasto **RESTORE**.

Proviamo ora a sviluppare un programma di animazione a colori, utilizzando a questo scopo anche i caratteri grafici predefiniti nella memoria ROM del computer, i quali consentono di raffigurare moduli e basi figurative per giochi, ecc.

Come già precisato nella lezione *Pratica del BASIC 1*, i caratteri grafici sono riportati sulla superficie anteriore della maggior parte dei tasti.

In particolare, i caratteri grafici posti sul lato destro dei tasti vengono visualizzati premendo contemporaneamente il tasto **SHIFT**, mentre quelli che si trovano sul lato sinistro dei tasti si visualizzano abbassando contemporaneamente il tasto **C=**.

Il programma in questione, il cui listato è riportato nella *fig. 12*, consente di simulare gli effetti di movimento di un piccolo aereo, naturalmente molto stilizzato, che si sposta sullo schermo da sinistra a destra, in concomitanza con le variazioni dei colori dello schermo stesso e del margine.

L'effetto del movimento viene ottenuto praticamente mediante la sostituzione veloce di un'immagine con un'altra eguale, utilizzando la funzione **TAB** (X) che, come è noto, consente di spostare l'immagine sullo schermo (nel nostro caso, da sinistra a destra), sempre però sulla stessa riga.

```
10 REM*ANIMAZIONE E COLORI*
12 PRINT"▯"
20 GOSUB1000
21 L1$="                                        "
22 POKE53280,1
24 FORSF=0TO15
26 FORCO=1TO15
27 POKE53280,SF
28 POKE53281,CO
30 FORX=1TO26
40 PRINT"▣"
42 FORY=1TO5
45 PRINTL1$
47 NEXTY
48 PRINT"▣"
55 PRINTTAB(X)"■          ▣▀■"
60 PRINTTAB(X)"■┌────────┐  ▣ ■"
70 PRINTTAB(X)"■| BASIC ├▣    ■─"
80 PRINTTAB(X)"■└────────┘  ▣ ■"
90 PRINTTAB(X)"■          ▛"
91 FORZ=1TO50:NEXTZ
100 NEXTX
130 NEXTCO
136 NEXTSF
140 PRINT"▯":END
1000 PRINT"▣▣▣▣▣▣▣▣▣▣"
1010 PRINT"■          ▣   ■     ▣   ■"
1020 PRINT"■          ▣   ■     ▣   ■     ▣   ■     ▣   ■"
1030 PRINT"■    ▣   ▣   ▣   ■     ▣     ■   ▣   ■   ▣   ■   ▣   ■"
1040 PRINT"■    ▣   ▣   ▣   ■   ▣         ■   ▣   ■   ▣   ■   ▣   ■"
1050 PRINT"■    ▣   ■   ▣   ■   ▣             ■   ▣   ■   ▣   ■"
1060 PRINT"■    ▣   ■   ▣   ■   ▣             ■   ▣   ■   ▣   ■"
1070 PRINT"■ ▣      ■   ▣   ■   ▣                 ■   ▣   ■"
1080 PRINT"■ ▣                                    "
1090 PRINT"■ ▣                                    "
1100 PRINT"■ ▣                                    "
1110 PRINT"■ ▣                                    "
1120 RETURN
```

Fig. 12 – Listato del programma "Animazione a colori" per il C64.

Analizziamo ora nelle sue linee fondamentali il listato del programma riportato nella *fig. 12*.

Il programma inizia con l'istruzione 12, che cancella lo schermo e posiziona il cursore in alto a sinistra.

L'istruzione 2Ø (**GOSUB**) provvede a fare saltare l'elaborazione al sotto-

programma (subroutine), che inizia con l'istruzione 1ØØØ e termina con l'istruzione 112Ø.

Le istruzioni di questo sottoprogramma stampano le stringhe per la composizione, nella parte inferiore dello schermo, del profilo delle case di una città, sulle quali "volerà" l'aereo (*fig. 13*).

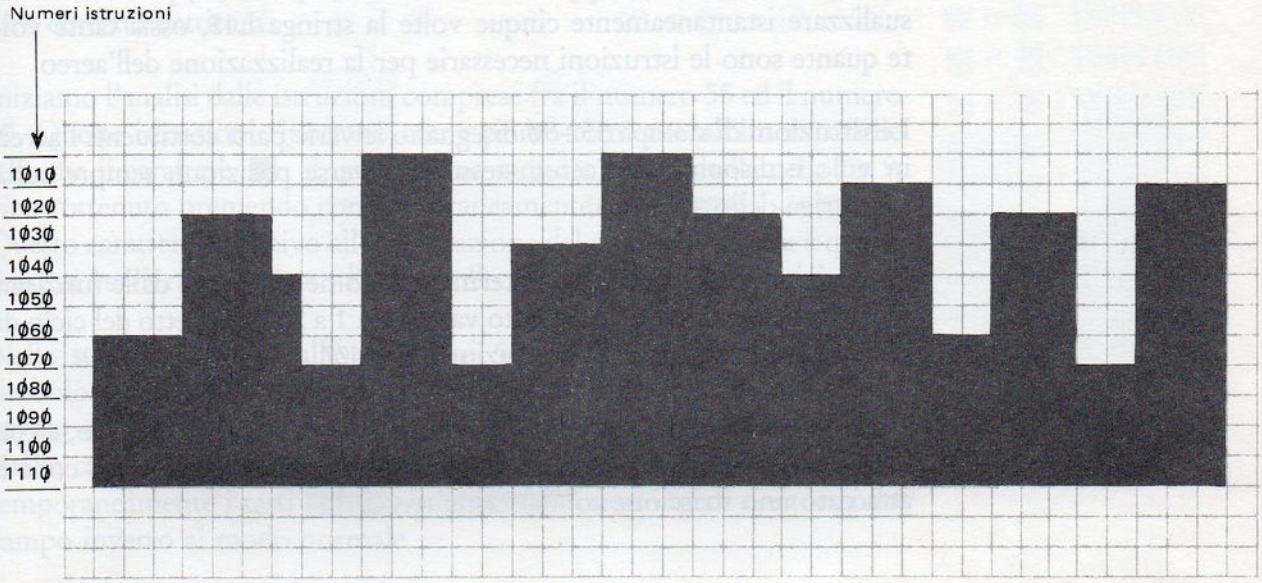


Fig. 13 – Profilo delle "case" visualizzato sullo schermo del televisore.

Come si può notare sulla stessa figura, alla sinistra del profilo delle case sono riportati i numeri delle corrispondenti istruzioni impiegate per la composizione dei diversi elementi che costituiscono il profilo stesso.

Dopo l'esecuzione dell'istruzione 112Ø (**RETURN**) il programma ritorna all'istruzione che segue il **GOSUB**, ossia all'istruzione 21.

Questa istruzione provvede ad assegnare alla variabile L 1$ una stringa costituita unicamente da 39 spazi bianchi, ottenuti battendo **esattamente** trentanove volte la barra spaziatrice, i quali definiscono la zona (rotta) entro la quale "volerà" l'aereo.

La barra spaziatrice viene frequentemente indicata in forma abbreviata con la lettera **b** sbarrata, ossia ƀ.

Pertanto, l'istruzione in questione può essere scritta piú chiaramente in

Pagina mancante

dell'aereo (55÷9Ø) e nelle istruzioni **PRINT** relative al disegno del profilo delle case (1Ø1Ø÷111Ø) sono presenti diversi spazi bianchi, che **occorre rispettare rigorosamente durante la battitura delle istruzioni stesse** per ottenere la corretta rappresentazione grafica prevista.

Esaminiamo ora piú dettagliatamente questi due gruppi di istruzioni, al fine di specificare esattamente il numero degli spazi presenti in ciascuna di esse e, nello stesso tempo, di fornire tutte le possibili indicazioni per la loro corretta battitura.

Iniziamo l'analisi dalle istruzioni comprese fra il numero 55 ed il numero 9Ø. Queste istruzioni riportano come primo elemento all'interno delle stringhe il carattere ■, il quale, come indicato nella tabella della *fig. 9*, viene ottenuto premendo contemporaneamente i tasti CTRL → 1. Questo carattere è relativo all'impostazione del colore nero, che è quello con cui verrà rappresentato l'aereo sullo schermo.

Nelle stringhe relative alle istruzioni 55÷8Ø sono inoltre presenti il simbolo grafico R, ottenuto premendo contemporaneamente i tasti CTRL → RVS | ON, con i quali si abilita la rappresentazione dei caratteri in campo inverso, ed il simbolo grafico ▬, ottenuto premendo contemporaneamente i tasti CTRL → RVS | OFF, con i quali si ritorna dal campo inverso al modo normale.

Di seguito sono riscritte le istruzioni 55÷9Ø, e per ciascuna di esse sono indicati chiaramente gli spazi previsti (ottenuti battendo la barra spaziatrice indicata con ♿) ed i diversi caratteri grafici predefiniti, utilizzati per la costruzione dell'aereo.

55 **PRINTTAB** (X) " ■ 9♿ R C= → * ▬ "
6Ø **PRINTTAB** (X) " ■ SHIFT → O C= → Y (5 volte) SHIFT → P 2♿ R 1♿ ▬ "
7Ø **PRINTTAB** (X) " ■ C= → H 1♿ BASIC C= → N SHIFT → C R 4♿ ▬ SHIFT → C "
8Ø **PRINTTAB** (X) " ■ SHIFT → L C= → P (5 volte) SHIFT → @ 2♿ R 1♿ ▬ "
9Ø **PRINTTAB** (X) " ■ 9♿ SHIFT → £ "

Vediamo ora come devono essere battute le istruzioni relative alla costruzione del profilo delle case (1ØØØ÷111Ø).

In particolare, nelle istruzioni 1Ø1Ø÷111Ø il primo elemento di ciascuna stringa è costituito dal simbolo relativo all'impostazione del colore nero (■), che è appunto quello utilizzato per la rappresentazione del grafico.

CONTROLLO DIDATTICO 7

Domande

1) - Illustrare le fasi operative delle istruzioni che costituiscono il seguente programma BASIC:

```
10 N1=2
15 N2=10
20 FOR X= N1 TO N2 STEP 2
25 PRINT X
30 NEXT X
35 END
```

.....

.....

.....

.....

.....

.....

.....

.....

2) - Illustrare gli effetti della seguente istruzione:

FOR X=Ø.5 TO -12 **STEP** -3

.....

.....

.....

3) - Con quali istruzioni BASIC si archiviano i dati all'interno di un programma?

.....

.....

4) - Quale tipo di organizzazione hanno i dati archiviati all'interno di un programma?

.....

.....

5) - Con quale istruzione BASIC è possibile immettere nella memoria del computer, mediante la tastiera, un carattere per volta?

.....

.....

6) - Per effettuare l'ordinamento di una successione di dati è piú conveniente usare l'istruzione **FOR... NEXT...** oppure l'istruzione **IF... THEN...**?

.....

.....

7) - Scrivere un semplice programma BASIC che conti quanti dati numerici sono presenti nella seguente successione:

-100, -95, -90,..., -15, -10, -5, Ø,5, 10, 15,..., 90, 95, 100

.....

.....

.....

.....

8) - Calcolare con un programma BASIC, che utilizza l'istruzione **FOR... NEXT...**, la media e la varianza della seguente serie di numeri:

0.5, 1, 2, -3, 1, 2, 44, 55, 2.3

da introdurre attraverso istruzioni **INPUT**. Per una piú agevole stesura di questo programma, rivedere l'esempio di calcolo della media e della varianza di una serie di numeri casuali presentato nella lezione di *Analisi e Programmazione 6*. Disegnare inoltre il relativo flow-chart.

Risposte

1) - Nelle istruzioni 1Ø e 15 vengono assegnati a N1 e N2 rispettivamente i numeri 2 e 1Ø, corrispondenti ai valori minimo e massimo della variabile di controllo X della procedura ciclica definita dalle istruzioni 2Ø e 3Ø. Ad ogni giro del ciclo, X assumerà i seguenti valori: 2, 4, 6, 8, 1Ø, i quali verranno visualizzati sullo schermo mediante l'istruzione 25.

2) - In questa istruzione, X assumerà, nei giri successivi del ciclo, i seguenti valori: Ø.5, –2.5, –5.5, – 8.5, –11.5.

3) - Le istruzioni BASIC che consentono l'archiviazione dei dati (numeri o stringhe) all'interno di un programma sono le cosiddette istruzioni-archivio **DATA**.

4) - L'organizzazione dei dati (numeri e stringhe) riportati ed elencati in istruzioni-archivio **DATA** è sequenziale. Per potere immettere questi dati nella memoria principale del computer occorre eseguire un'istruzione di lettura **READ**, seguita da un elenco di nomi di variabili. Questi nomi devono rispettare la corrispondenza tipologica dei valori immessi in memoria.

5) - L'istruzione BASIC che consente di introdurre nella memoria del computer un solo carattere per volta è l'istruzione **GET** D$, dove **GET** è la parola chiave dell'istruzione e D$ è il nome di una variabile stringa, a cui va assegnato il valore del carattere introdotto.

6) - Per ordinare una successione di dati è piú indicata l'istruzione di diramazione condizionale **IF... THEN...**, perché quest'ultima è specificamente adatta per eseguire il confronto logico tra due valori (numerico o stringa), confronto che è alla base di un qualsiasi algoritmo di ordinamento.

7) - La successione dei numeri:

–1ØØ, –95, –9Ø, ..., –15, –1Ø, –5, Ø,5, 1Ø, 15, ..., 9Ø, 95, 1ØØ

consta esattamente di 41 numeri (venti compresi tra 5 e 1ØØ, altri venti compresi fra –1ØØ e –5, ed il numero Ø che costituisce il punto centrale della successione).

Per calcolare il numero dei dati della successione sopra riportata è sufficiente eseguire il seguente, breve programma:

```
10 N=0
15 FOR I=-100 TO 100 STEP 5
20 N=N+1
25 NEXT I
30 PRINT N
```

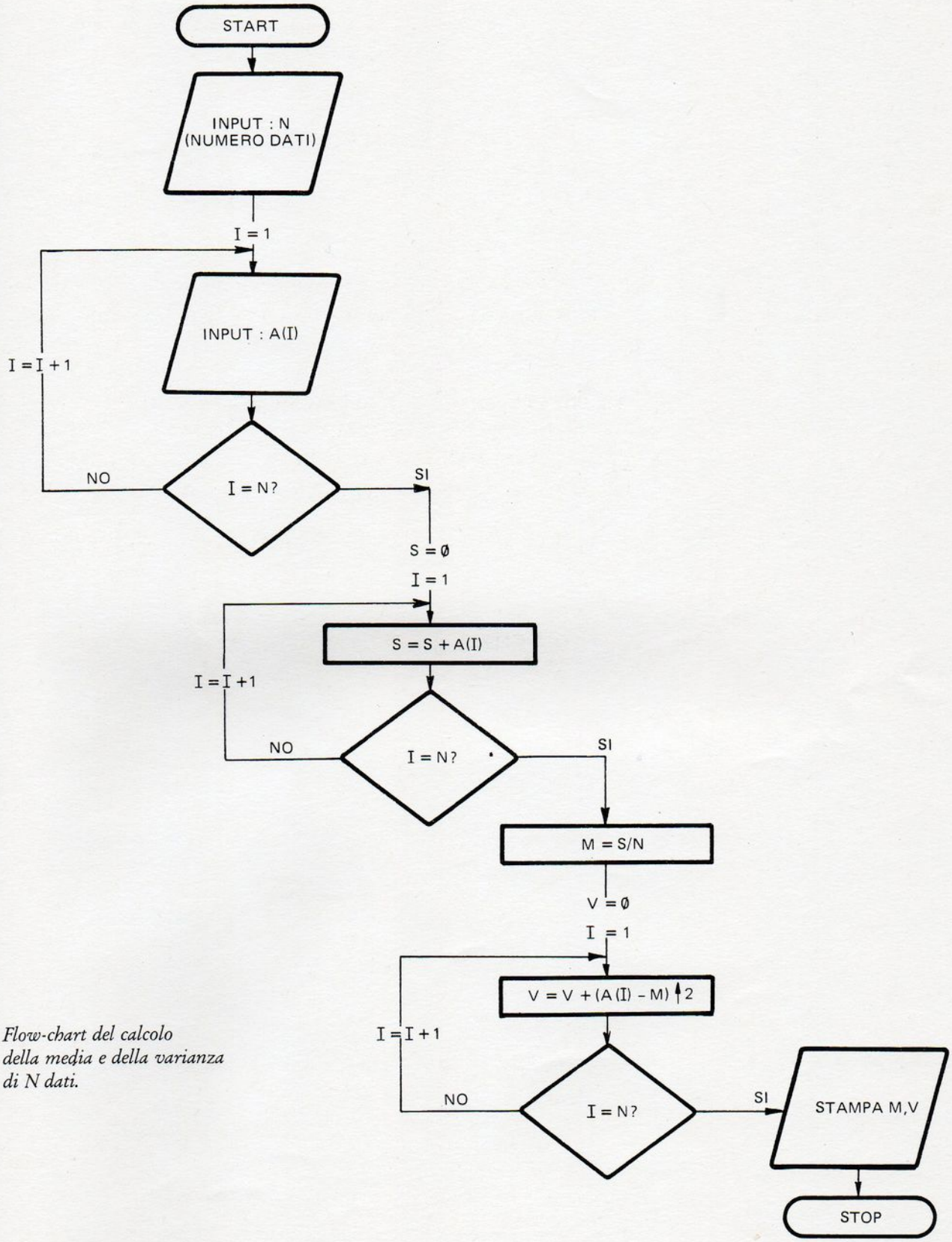
Nel programma in questione la variabile di nome N serve a contare appunto il numero degli interi compresi nella succcessione considerata.

8) - Di seguito sono riportati la codifica in BASIC e il flow-chart di uno dei possibili criteri di programmazione della media e della varianza di un numero qualsiasi N di dati.
Per la serie di numeri seguente: Ø.5, 1, 2, –3, 1, 2, 44, 55, 2.3 si avrà:

MEDIA = 11.6444444 VARIANZA = 3765.2Ø223

```
10 REM*MEDIA E VARIANZA*
12 REM*   DI N. DATI   *
15 INPUT"INTR.N.DATI";N
20 DIM A(N)
25 FOR I=1 TO N
30 INPUT"INTR.DATI";A(I)
35 NEXT I
40 S=0
45 FOR I=1  TO N
50 S=S+A(I)
60 NEXT I
65 M=S/N
70 V=0
75 FOR I=1 TO N
80 V=V+(A(I)-M)↑2
85 NEXT I
90 PRINT"Ø"
95 PRINT"LA MEDIA E LA VARIANZA"
100 PRINT"DEI";N;"DATI E'"
105 PRINT
110 PRINT M,V
115 END
```

Listato BASIC del calcolo della media e della varianza di un numero N di dati.



Flow-chart del calcolo della media e della varianza di N dati.

PARLA BASIC

Corso teorico-pratico per corrispondenza di

ANALISI E PROGRAMMAZIONE BASIC

Scuola Radio Elettra

10126 Torino - Via Stellone 5

Tel. (011) 674432